

ETSI TS 129 198-3 V7.1.0 (2007-06)

Technical Specification

**Universal Mobile Telecommunications System (UMTS);
Open Service Access (OSA)
Application Programming Interface (API);
Part 3: Framework
(3GPP TS 29.198-03 version 7.1.0 Release 7)**



Reference

RTS/TSGC-0529198-03v710

Keywords

UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:

<http://www.etsi.org>

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at

<http://portal.etsi.org/tb/status/status.asp>

If you find errors in the present document, please send your comment to one of the following services:

http://portal.etsi.org/chaicor/ETSI_support.asp

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2007.
All rights reserved.

DECTTM, **PLUGTESTS**TM and **UMTS**TM are Trade Marks of ETSI registered for the benefit of its Members.
TIPHONTM and the **TIPHON logo** are Trade Marks currently being registered by ETSI for the benefit of its Members.
3GPPTM is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<http://webapp.etsi.org/IPR/home.asp>).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under <http://webapp.etsi.org/key/queryform.asp>.

Contents

Intellectual Property Rights	2
Foreword.....	2
Foreword.....	12
Introduction	12
1 Scope	14
2 References	14
3 Definitions, symbols and abbreviations	15
3.1 Definitions	15
3.2 Abbreviations	15
4 Overview of the Framework.....	15
5 The Base Interface Specification.....	16
5.1 Interface Specification Format	16
5.1.1 Interface Class	16
5.1.2 Method descriptions.....	17
5.1.3 Parameter descriptions.....	17
5.1.4 State Model.....	17
5.2 Base Interface	17
5.2.1 Interface Class IpInterface	17
5.3 Service Interfaces	17
5.3.1 Overview	17
5.4 Generic Service Interface	17
5.4.1 Interface Class IpService	17
5.4.1.1 Method setCallback()	18
5.4.1.2 Method setCallbackWithSessionID().....	18
6 Framework Access Session API.....	19
6.1 Sequence Diagrams	19
6.1.1 Trust and Security Management Sequence Diagrams	19
6.1.1.1 Initial Access.....	19
6.1.1.2 Framework Terminates Access	20
6.1.1.3 Application Terminates Access.....	21
6.1.1.4 Non-API level Authentication.....	22
6.1.1.5 API Level Authentication	23
6.2 Class Diagrams.....	24
6.3 Interface Classes.....	25
6.3.1 Trust and Security Management Interface Classes	25
6.3.1.1 Interface Class IpClientAPILevelAuthentication.....	25
6.3.1.1.1 Method abortAuthentication()	26
6.3.1.1.2 Method authenticationSucceeded()	26
6.3.1.1.3 Method challenge().....	26
6.3.1.2 Interface Class IpClientAccess.....	27
6.3.1.2.1 Method terminateAccess().....	28
6.3.1.3 Interface Class IpInitial	28
6.3.1.3.1 Method initiateAuthenticationWithVersion().....	29
6.3.1.4 Interface Class IpAuthentication	30
6.3.1.4.1 Method requestAccess()	30
6.3.1.5 Interface Class IpAPILevelAuthentication	31
6.3.1.5.1 Method abortAuthentication()	31
6.3.1.5.2 Method authenticationSucceeded()	32
6.3.1.5.3 Method selectAuthenticationMechanism().....	32
6.3.1.5.4 Method challenge().....	32
6.3.1.6 Interface Class IpAccess	34

6.3.1.6.1	Method obtainInterface()	34
6.3.1.6.2	Method obtainInterfaceWithCallback()	34
6.3.1.6.3	Method listInterfaces()	35
6.3.1.6.4	Method selectSigningAlgorithm()	35
6.3.1.6.5	Method terminateAccess()	36
6.3.1.6.6	Method relinquishInterface()	36
6.4	State Transition Diagrams	37
6.4.1	Trust and Security Management State Transition Diagrams	37
6.4.1.1	State Transition Diagrams for IpInitial	37
6.4.1.2	State Transition Diagrams for IpAPILevelAuthentication	37
6.4.1.2.1	Idle State	38
6.4.1.2.2	Authenticating Framework State	38
6.4.1.2.3	Framework Authenticated State	39
6.4.1.2.4	Authenticating Client State	39
6.4.1.2.5	Client Authenticated State	39
6.4.1.2.6	Idle State	40
6.4.1.2.7	Authenticating Framework State	40
6.4.1.2.8	Framework Authenticated State	40
6.4.1.2.9	Authenticating Client State	41
6.4.1.2.10	Client Authenticated State	41
6.4.1.3	State Transition Diagrams for IpAccess	41
6.4.1.3.1	Active State	42
7	Framework-to-Application API	42
7.1	Sequence Diagrams	42
7.1.1	Event Notification Sequence Diagrams	42
7.1.1.1	Enable Event Notification	42
7.1.2	Integrity Management Sequence Diagrams	43
7.1.2.1	Load Management: Suspend/resume notification from application	43
7.1.2.2	Load Management: Framework queries load statistics	44
7.1.2.3	Load Management: Framework callback registration and Application load control	45
7.1.2.4	Load Management: Application reports current load condition	46
7.1.2.5	Load Management: Application queries load statistics	47
7.1.2.6	Load Management: Application callback registration and load control	48
7.1.2.7	Heartbeat Management: Start/perform/end heartbeat supervision of the application	49
7.1.2.8	Fault Management: Framework detects a Service failure	50
7.1.2.9	Fault Management: Application requests a Framework activity test	51
7.1.3	Service Discovery Sequence Diagrams	52
7.1.3.1	Service Discovery	52
7.1.4	Service Agreement Management Sequence Diagrams	54
7.1.4.1	Service Selection	54
7.2	Class Diagrams	56
7.3	Interface Classes	59
7.3.1	Service Discovery Interface Classes	59
7.3.1.1	Interface Class IpServiceDiscovery	59
7.3.1.1.1	Method listServiceTypes()	60
7.3.1.1.2	Method describeServiceType()	60
7.3.1.1.3	Method discoverService()	61
7.3.1.1.4	Method listSubscribedServices()	62
7.3.2	Service Agreement Management Interface Classes	62
7.3.2.1	Interface Class IpAppServiceAgreementManagement	62
7.3.2.1.1	Method signServiceAgreement()	63
7.3.2.1.2	Method terminateServiceAgreement()	63
7.3.2.2	Interface Class IpServiceAgreementManagement	64
7.3.2.2.1	Method signServiceAgreement()	64
7.3.2.2.2	Method terminateServiceAgreement()	65
7.3.2.2.3	Method selectService()	66
7.3.2.2.4	Method initiateSignServiceAgreement()	66
7.3.3	Integrity Management Interface Classes	67
7.3.3.1	Interface Class IpAppFaultManager	67
7.3.3.1.1	Method activityTestRes()	67
7.3.3.1.2	Method appActivityTestReq()	68

7.3.3.1.3	Method <<deprecated>> fwFaultReportInd()	68
7.3.3.1.4	Method <<deprecated>> fwFaultRecoveryInd()	68
7.3.3.1.5	Method <<deprecated>> fwUnavailableInd()	68
7.3.3.1.6	Method activityTestErr()	69
7.3.3.1.7	Method appUnavailableInd()	69
7.3.3.1.8	Method svcAvailStatusInd()	69
7.3.3.1.9	Method generateFaultStatisticsRecordRes()	69
7.3.3.1.10	Method generateFaultStatisticsRecordErr()	70
7.3.3.1.11	Method generateFaultStatisticsRecordReq()	70
7.3.3.1.12	Method fwAvailStatusInd()	70
7.3.3.2	Interface Class IpFaultManager	71
7.3.3.2.1	Method activityTestReq()	71
7.3.3.2.2	Method appActivityTestRes()	72
7.3.3.2.3	Method svcUnavailableInd()	72
7.3.3.2.4	Method appActivityTestErr()	72
7.3.3.2.5	Method appAvailStatusInd()	73
7.3.3.2.6	Method generateFaultStatisticsRecordReq()	73
7.3.3.2.7	Method generateFaultStatisticsRecordRes()	74
7.3.3.2.8	Method generateFaultStatisticsRecordErr()	74
7.3.3.3	Interface Class IpAppHeartBeatMgmt	74
7.3.3.3.1	Method enableAppHeartBeat()	75
7.3.3.3.2	Method disableAppHeartBeat()	75
7.3.3.3.3	Method changeInterval()	75
7.3.3.4	Interface Class IpAppHeartBeat	75
7.3.3.4.1	Method pulse()	76
7.3.3.5	Interface Class IpHeartBeatMgmt	76
7.3.3.5.1	Method enableHeartBeat()	76
7.3.3.5.2	Method disableHeartBeat()	77
7.3.3.5.3	Method changeInterval()	77
7.3.3.6	Interface Class IpHeartBeat	77
7.3.3.6.1	Method pulse()	77
7.3.3.7	Interface Class IpAppLoadManager	78
7.3.3.7.1	Method loadLevelNotification()	78
7.3.3.7.2	Method resumeNotification()	78
7.3.3.7.3	Method suspendNotification()	79
7.3.3.7.4	Method createLoadLevelNotification()	79
7.3.3.7.5	Method destroyLoadLevelNotification()	79
7.3.3.7.6	Method queryAppLoadStatsReq()	79
7.3.3.7.7	Method queryLoadStatsRes()	79
7.3.3.7.8	Method queryLoadStatsErr()	80
7.3.3.8	Interface Class IpLoadManager	80
7.3.3.8.1	Method reportLoad()	81
7.3.3.8.2	Method createLoadLevelNotification()	81
7.3.3.8.3	Method destroyLoadLevelNotification()	82
7.3.3.8.4	Method resumeNotification()	82
7.3.3.8.5	Method suspendNotification()	82
7.3.3.8.6	Method queryLoadStatsReq()	83
7.3.3.8.7	Method queryAppLoadStatsRes()	83
7.3.3.8.8	Method queryAppLoadStatsErr()	84
7.3.3.9	Interface Class IpOAM	84
7.3.3.9.1	Method systemDateTimeQuery()	84
7.3.3.10	Interface Class IpAppOAM	85
7.3.3.10.1	Method systemDateTimeQuery()	85
7.3.4	Event Notification Interface Classes	85
7.3.4.1	Interface Class IpAppEventNotification	85
7.3.4.1.1	Method reportNotification()	86
7.3.4.1.2	Method notificationTerminated()	86
7.3.4.2	Interface Class IpEventNotification	86
7.3.4.2.1	Method createNotification()	87
7.3.4.2.2	Method destroyNotification()	87
7.4	State Transition Diagrams	87
7.4.1	Service Discovery State Transition Diagrams	87

7.4.1.1	State Transition Diagrams for IpServiceDiscovery.....	87
7.4.1.1.1	Active State	88
7.4.2	Service Agreement Management State Transition Diagrams	88
7.4.3	Integrity Management State Transition Diagrams	88
7.4.3.1	State Transition Diagrams for IpLoadManager.....	88
7.4.3.1.1	Idle State	89
7.4.3.1.2	Notification Suspended State.....	89
7.4.3.1.3	Active State	89
7.4.3.2	State Transition Diagrams for LoadManagerInternal.....	89
7.4.3.2.1	Normal load State	90
7.4.3.2.2	Application Overload State	90
7.4.3.2.3	Internal overload State.....	90
7.4.3.2.4	Internal and Application Overload State	90
7.4.3.3	State Transition Diagrams for IpOAM.....	91
7.4.3.3.1	Active State	91
7.4.3.4	State Transition Diagrams for IpFaultManager.....	91
7.4.3.4.1	Framework Active State	92
7.4.3.4.2	Framework Faulty State.....	92
7.4.3.4.3	Framework Activity Test State.....	92
7.4.3.4.4	Service Activity Test State	92
7.4.4	Event Notification State Transition Diagrams	92
7.4.4.1	State Transition Diagrams for IpEventNotification	92
8	Framework-to-Service API	93
8.1	Sequence Diagrams	93
8.1.1	Service Discovery Sequence Diagrams	93
8.1.2	Service Registration Sequence Diagrams	93
8.1.2.1	New SCF Sub Type Registration	93
8.1.2.2	New SCF Registration.....	94
8.1.3	Service Instance Lifecycle Manager Sequence Diagrams	96
8.1.3.1	Sign Service Agreement.....	96
8.1.4	Integrity Management Sequence Diagrams	97
8.1.4.1	Load Management: Service callback registration and load control.....	97
8.1.4.2	Load Management: Framework callback registration and service load control.....	98
8.1.4.3	Load Management: Client and Service Load Balancing.....	99
8.1.4.4	Heartbeat Management: Start/perform/end heartbeat supervision of the service	100
8.1.4.5	Fault Management: Service requests Framework activity test	101
8.1.4.6	Fault Management: Service requests Application activity test	102
8.1.4.7	Fault Management: Application requests Service activity test	102
8.1.4.8	Fault Management: Application detects service is unavailable.....	104
8.1.5	Event Notification Sequence Diagrams	104
8.2	Class Diagrams.....	104
8.3	Interface Classes.....	108
8.3.1	Service Registration Interface Classes	108
8.3.1.1	Interface Class IpFwServiceRegistration	108
8.3.1.1.1	Method registerService()	108
8.3.1.1.2	Method announceServiceAvailability().....	109
8.3.1.1.3	Method unregisterService()	110
8.3.1.1.4	Method describeService().....	110
8.3.1.1.5	Method unannounceService().....	111
8.3.1.1.6	Method registerServiceSubType().....	111
8.3.2	Service Instance Lifecycle Manager Interface Classes	112
8.3.2.1	Interface Class IpServiceInstanceLifecycleManager	112
8.3.2.1.1	Method createServiceManager()	113
8.3.2.1.2	Method destroyServiceManager()	113
8.3.3	Service Discovery Interface Classes	114
8.3.3.1	Interface Class IpFwServiceDiscovery	114
8.3.3.1.1	Method listServiceTypes()	114
8.3.3.1.2	Method describeServiceType().....	115
8.3.3.1.3	Method discoverService().....	115
8.3.3.1.4	Method listRegisteredServices().....	116
8.3.4	Integrity Management Interface Classes.....	116

8.3.4.1	Interface Class IpFwFaultManager	116
8.3.4.1.1	Method activityTestReq()	117
8.3.4.1.2	Method svcActivityTestRes()	118
8.3.4.1.3	Method appUnavailableInd()	118
8.3.4.1.4	Method svcActivityTestErr()	118
8.3.4.1.5	Method svcAvailStatusInd()	119
8.3.4.1.6	Method generateFaultStatisticsRecordReq()	119
8.3.4.1.7	Method generateFaultStatisticsRecordRes()	119
8.3.4.1.8	Method generateFaultStatisticsRecordErr()	120
8.3.4.2	Interface Class IpSvcFaultManager	120
8.3.4.2.1	Method activityTestRes()	121
8.3.4.2.2	Method svcActivityTestReq()	121
8.3.4.2.3	Method <<deprecated>> fwFaultReportInd()	122
8.3.4.2.4	Method <<deprecated>> fwFaultRecoveryInd()	122
8.3.4.2.5	Method <<deprecated>> fwUnavailableInd()	123
8.3.4.2.6	Method svcUnavailableInd()	123
8.3.4.2.7	Method activityTestErr()	123
8.3.4.2.8	Method appAvailStatusInd()	123
8.3.4.2.9	Method generateFaultStatisticsRecordRes()	124
8.3.4.2.10	Method generateFaultStatisticsRecordErr()	124
8.3.4.2.11	Method generateFaultStatisticsRecordReq()	125
8.3.4.2.12	Method fwAvailStatusInd()	125
8.3.4.3	Interface Class IpFwHeartBeatMgmt	125
8.3.4.3.1	Method enableHeartBeat()	126
8.3.4.3.2	Method disableHeartBeat()	126
8.3.4.3.3	Method changeInterval()	126
8.3.4.4	Interface Class IpFwHeartBeat	127
8.3.4.4.1	Method pulse()	127
8.3.4.5	Interface Class IpSvcHeartBeatMgmt	127
8.3.4.5.1	Method enableSvcHeartBeat()	128
8.3.4.5.2	Method disableSvcHeartBeat()	128
8.3.4.5.3	Method changeInterval()	128
8.3.4.6	Interface Class IpSvcHeartBeat	129
8.3.4.6.1	Method pulse()	129
8.3.4.7	Interface Class IpFwLoadManager	129
8.3.4.7.1	Method reportLoad()	130
8.3.4.7.2	Method createLoadLevelNotification()	131
8.3.4.7.3	Method destroyLoadLevelNotification()	131
8.3.4.7.4	Method suspendNotification()	131
8.3.4.7.5	Method resumeNotification()	132
8.3.4.7.6	Method queryLoadStatsReq()	132
8.3.4.7.7	Method querySvcLoadStatsRes()	132
8.3.4.7.8	Method querySvcLoadStatsErr()	133
8.3.4.8	Interface Class IpSvcLoadManager	133
8.3.4.8.1	Method loadLevelNotification()	134
8.3.4.8.2	Method suspendNotification()	134
8.3.4.8.3	Method resumeNotification()	134
8.3.4.8.4	Method createLoadLevelNotification()	135
8.3.4.8.5	Method destroyLoadLevelNotification()	135
8.3.4.8.6	Method querySvcLoadStatsReq()	135
8.3.4.8.7	Method queryLoadStatsRes()	135
8.3.4.8.8	Method queryLoadStatsErr()	136
8.3.4.9	Interface Class IpFwOAM	136
8.3.4.9.1	Method systemDateTimeQuery()	136
8.3.4.10	Interface Class IpSvcOAM	137
8.3.4.10.1	Method systemDateTimeQuery()	137
8.3.5	Event Notification Interface Classes	138
8.3.5.1	Interface Class IpFwEventNotification	138
8.3.5.1.1	Method createNotification()	138
8.3.5.1.2	Method destroyNotification()	138
8.3.5.2	Interface Class IpSvcEventNotification	139
8.3.5.2.1	Method reportNotification()	139

8.3.5.2.2	Method notificationTerminated()	140
8.4	State Transition Diagrams	140
8.4.1	Service Registration State Transition Diagrams	140
8.4.1.1	State Transition Diagrams for IpFwServiceRegistration	140
8.4.1.1.1	SCF Registered State	141
8.4.1.1.2	SCF Announced State	141
8.4.2	Service Instance Lifecycle Manager State Transition Diagrams	141
8.4.3	Service Discovery State Transition Diagrams	141
8.4.4	Integrity Management State Transition Diagrams	142
8.4.4.1	State Transition Diagrams for IpFwLoadManager	142
8.4.4.1.1	Idle State	142
8.4.4.1.2	Notification Suspended State	142
8.4.4.1.3	Active State	142
8.4.4.2	State Transition Diagrams for IpFwFaultManager	143
8.4.4.2.1	Framework Active State	143
8.4.4.2.2	Framework Activity Test State	143
8.4.4.2.3	Application Activity Test State	143
8.4.4.2.4	Framework Faulty State	143
8.4.5	Event Notification State Transition Diagrams	143
9	Service Properties	144
9.1	Service Super and Sub Types	144
9.2	Service Property Types	144
9.3	General Service Properties	146
9.3.1	Service Name	147
9.3.2	Service Version	147
9.3.3	Service ID	147
9.3.4	Service Description	147
9.3.5	Product Name	147
9.3.6	Product Version	147
9.3.7	Void	148
9.3.8	Operation Set	148
9.3.9	Compatible Service	148
9.3.10	Backward Compatibility Level	149
9.3.11	Migration Required	150
9.3.12	Data Migrated	150
9.3.13	Migration Date And Time	151
9.3.14	Support for Regular Expressions in Address Range	151
10	Data Definitions	151
10.1	Common Framework Data Definitions	152
10.1.1	TpClientAppID	152
10.1.2	TpClientAppIDList	152
10.1.3	TpDomainID	152
10.1.4	TpDomainIDType	152
10.1.5	TpEntOpID	152
10.1.6	TpPropertyName	153
10.1.7	TpPropertyValue	153
10.1.8	TpProperty	153
10.1.9	TpPropertyList	153
10.1.10	TpEntOpIDList	153
10.1.11	TpFwID	153
10.1.12	TpService	153
10.1.13	TpServiceList	153
10.1.14	TpServiceDescription	153
10.1.15	TpServiceID	154
10.1.16	TpServiceIDList	154
10.1.17	TpServiceInstanceID	154
10.1.18	TpServiceTypeProperty	154
10.1.19	TpServiceTypePropertyList	154
10.1.20	TpServiceTypePropertyMode	154
10.1.21	TpServicePropertyTypeName	154

10.1.22	TpServicePropertyName	154
10.1.23	TpServicePropertyNameList	155
10.1.24	TpServicePropertyValue	155
10.1.25	TpServicePropertyValueList	155
10.1.26	TpServiceProperty	155
10.1.27	TpServicePropertyList	155
10.1.28	TpServiceSupplierID	155
10.1.29	TpServiceTypeDescription	155
10.1.30	TpServiceTypeName	156
10.1.31	TpServiceTypeNameList	156
10.1.32	TpSubjectType	156
10.1.33	TpServiceTypePropertyValue	157
10.1.34	TpServiceTypePropertyValueList	157
10.2	Event Notification Data Definitions	157
10.2.1	TpFwEventName	157
10.2.2	TpFwEventCriteria	158
10.2.3	TpFwEventInfo	158
10.2.4	TpFwMigrationServiceAvailableInfo	158
10.2.5	TpMigrationAdditionalInfo	159
10.2.6	TpMigrationAdditionalInfoType	159
10.2.7	TpMigrationAdditionalInfoSet	160
10.2.8	TpFwAgreementInfo	160
10.3	Trust and Security Management Data Definitions	160
10.3.1	TpAccessType	160
10.3.2	TpAuthType	160
10.3.3	Void	161
10.3.4	Void	161
10.3.5	Void	161
10.3.6	TpAuthDomain	161
10.3.7	TpInterfaceName	161
10.3.8	TpInterfaceNameList	161
10.3.9	TpServiceToken	161
10.3.10	TpSignatureAndServiceMgr	162
10.3.11	TpSigningAlgorithm	162
10.3.12	TpSigningAlgorithmCapabilityList	162
10.3.13	TpAuthMechanism	163
10.3.14	TpAuthMechanismList	163
10.4	Integrity Management Data Definitions	163
10.4.1	TpActivityTestRes	163
10.4.2	TpFaultStatsRecord	163
10.4.3	TpFaultStats	163
10.4.4	TpFaultStatisticsError	164
10.4.5	TpFaultStatsSet	164
10.4.6	TpActivityTestID	164
10.4.7	TpInterfaceFault	164
10.4.8	Void	164
10.4.9	TpFwUnavailReason	164
10.4.10	TpLoadLevel	164
10.4.11	TpLoadThreshold	165
10.4.12	TpLoadInitVal	165
10.4.13	TpLoadPolicy	165
10.4.14	TpLoadStatistic	165
10.4.15	TpLoadStatisticList	165
10.4.16	TpLoadStatisticData	165
10.4.17	TpLoadStatisticEntityID	166
10.4.18	TpLoadStatisticEntityType	166
10.4.19	TpLoadStatisticInfo	166
10.4.20	TpLoadStatisticInfoType	166
10.4.21	TpLoadStatisticError	166
10.4.22	TpSvcAvailStatusReason	167
10.4.23	TpAppAvailStatusReason	167
10.4.24	TpLoadTestID	167

10.4.25	TpFaultStatsErrorList	167
10.4.26	TpFaultReqID	167
10.4.27	TpFwAvailStatusReason	168
10.5	Service Subscription Data Definitions	168
10.5.1	TpPropertyName.....	168
10.5.2	TpPropertyValue.....	168
10.5.3	TpProperty	168
10.5.4	TpPropertyList	168
10.5.5	TpEntOpProperties	168
10.5.6	TpEntOp	169
10.5.7	TpServiceContractID	169
10.5.8	TpServiceContractIDList.....	169
10.5.9	TpPersonName	169
10.5.10	TpPostalAddress	169
10.5.11	TpTelephoneNumber	169
10.5.12	TpEmail	169
10.5.13	TpHomePage	169
10.5.14	TpPersonProperties	169
10.5.15	TpPerson.....	170
10.5.16	TpServiceStartDate	170
10.5.17	TpServiceEndDate	170
10.5.18	TpServiceRequestor.....	170
10.5.19	TpBillingContact	170
10.5.20	TpServiceSubscriptionProperties.....	170
10.5.21	TpServiceContract	170
10.5.22	TpServiceContractDescription.....	171
10.5.23	TpClientAppProperties	171
10.5.24	TpClientAppDescription.....	171
10.5.25	TpSagID.....	171
10.5.26	TpSagIDList	171
10.5.27	TpSagDescription	171
10.5.28	TpSag.....	172
10.5.29	TpServiceProfileID	172
10.5.30	TpServiceProfileIDList.....	172
10.5.31	TpServiceProfile	172
10.5.32	TpServiceProfileDescription.....	173
10.5.33	TpSagProfilePair.....	173
10.5.34	TpAddSagMembersConflict	173
10.5.35	TpAddSagMembersConflictList.....	174
10.5.36	TpAssignSagToServiceProfileConflict.....	174
10.5.37	TpAssignSagToServiceProfileConflictList	174
11	Exception Classes.....	175
Annex A (normative):	OMG IDL Description of Framework	176
Annex B (informative):	W3C WSDL Description of Framework.....	177
Annex C (informative):	Java™ API Description of the Framework.....	178
Annex D (informative):	Description of the Framework for 3GPP2 cdma2000 networks.....	179
D.1	General Exceptions.....	179
D.2	Specific Exceptions	179
D.2.1	Clause 1: Scope	179
D.2.2	Clause 2: References	179
D.2.3	Clause 3: Definitions and abbreviations	179
D.2.4	Clause 4: Overview of the Framework.....	179
D.2.5	Clause 5: The Base Interface Specification	179
D.2.6	Clause 6: Framework Access Session API.....	180
D.2.7	Clause 7 Framework-to-Application Sequence Diagrams.....	180
D.2.8	Clause 8: Framework-to-Service API.....	180

D.2.9	Clause 9: Service Properties.....	180
D.2.10	Clause 10: Data Definitions.....	180
D.2.11	Clause 11: Exception Classes.....	180
D.2.12	Annex A (normative): OMG IDL Description of the Framework.....	180
D.2.13	Annex B (informative): W3C WSDL Description of the Framework.....	180
D.2.14	Annex C (informative): Java™ API Description of the Framework.....	180
Annex E (informative): Change history		181
History		182

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
 - 1 presented to TSG for information;
 - 2 presented to TSG for approval;
 - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

Introduction

The present document is part 3 of a multi-part TS covering the 3rd Generation Partnership Project: Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API), as identified below. The **API specification** (3GPP TS 29.198) is structured in the following Parts:

Part 1:	"Overview";	
Part 2:	"Common Data Definitions";	
Part 3:	"Framework";	
Part 4:	"Call Control";	
	Sub-part 1: "Call Control Common Definitions";	
	Sub-part 2: "Generic Call Control SCF";	
	Sub-part 3: "Multi-Party Call Control SCF";	
	Sub-part 4: "Multi-Media Call Control SCF";	
	Sub-part 5: "Conference Call Control SCF";	(not part of 3GPP Release 7)
Part 5:	"User Interaction SCF";	
Part 6:	"Mobility SCF";	
Part 7:	"Terminal Capabilities SCF";	
Part 8:	"Data Session Control SCF";	
Part 9:	"Generic Messaging SCF";	(not part of 3GPP Release 7)
Part 10:	"Connectivity Manager SCF";	(not part of 3GPP Release 7)
Part 11:	"Account Management SCF";	
Part 12:	"Charging SCF".	
Part 13:	"Policy Management SCF";	
Part 14:	"Presence and Availability Management SCF";	
Part 15:	"Multi Media Messaging SCF";	
Part 16:	"Service Broker SCF".	(new in 3GPP Release 7)

The **Mapping specification of the OSA APIs and network protocols** (3GPP TR 29.998) is also structured as above. A mapping to network protocols is however not applicable for all Parts, but the numbering of Parts is kept. Also in case a Part is not supported in a Release, the numbering of the parts is maintained.

Table: Overview of the OSA APIs & Protocol Mappings 29.198 & 29.998-family

OSA API specifications 29.198-family					OSA API Mapping - 29.998-family	
29.198-01	Overview				29.998-01	Overview
29.198-02	Common Data Definitions				29.998-02	<i>Not Applicable</i>
29.198-03	Framework				29.998-03	<i>Not Applicable</i>
Call Control (CC) SCF	29.198-04-1	29.198-04-2	29.198-04-3	29.198-04-4	29.998-04-1	Generic Call Control – CAP mapping
	Common CC data definitions	Generic CC SCF	Multi-Party CC SCF	Multi-media CC SCF	29.998-04-2	<i>Generic Call Control – INAP mapping</i>
					29.998-04-3	<i>Generic Call Control – Megaco mapping</i>
					29.998-04-4	Multiparty Call Control – ISC mapping
29.198-05	User Interaction SCF				29.998-05-1	User Interaction – CAP mapping
					29.998-05-2	<i>User Interaction – INAP mapping</i>
					29.998-05-3	<i>User Interaction – Megaco mapping</i>
					29.998-05-4	User Interaction – SMS mapping
29.198-06	Mobility SCF				29.998-06	User Status and User Location – MAP mapping
29.198-07	Terminal Capabilities SCF				29.998-07	<i>Not Applicable</i>
29.198-08	Data Session Control SCF				29.998-08	Data Session Control – CAP mapping
29.198-09	<i>Generic Messaging SCF</i>				29.998-09	<i>Not Applicable</i>
29.198-10	<i>Connectivity Manager SCF</i>				29.998-10	<i>Not Applicable</i>
29.198-11	Account Management SCF				29.998-11	<i>Not Applicable</i>
29.198-12	Charging SCF				29.998-12	<i>Not Applicable</i>
29.198-13	Policy Management SCF				29.998-13	<i>Not Applicable</i>
29.198-14	Presence & Availability Management SCF				29.998-14	<i>Not Applicable</i>
29.198-15	Multi Media Messaging SCF				29.998-15	<i>Not Applicable</i>
29.198-16	Service Broker SCF				29.998-16	<i>Not Applicable</i>

1 Scope

The present document is Part 3 of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA).

The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA APIs. The concepts and the functional architecture for the OSA are contained in 3GPP TS 23.198 [3]. The requirements for OSA are contained in 3GPP TS 22.127 [2].

The present document specifies the Framework aspects of the interface. All aspects of the Framework are defined in the present document, these being:

- Sequence Diagrams;
- Class Diagrams;
- Interface specification plus detailed method descriptions;
- State Transition diagrams;
- Data definitions;
- IDL Description of the interfaces.
- WSDL Description of the interfaces
- Reference to the Java™ API description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modelling Language (UML).

This specification has been defined jointly between 3GPP TSG CT WG5, ETSI TISPAN and The Parlay Group, in co-operation with a number of JAIN™ Community member companies.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

- [1] 3GPP TS 29.198-1: "Open Service Access; Application Programming Interface; Part 1: Overview".
- [2] 3GPP TS 22.127: "Service Requirement for the Open Services Access (OSA); Stage 1".
- [3] 3GPP TS 23.198: "Open Service Access (OSA); Stage 2".
- [4] IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August 1996].

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TS 29.198-1 [1] apply.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TS 29.198-1 [1] apply.

4 Overview of the Framework

This clause explains which basic mechanisms are executed in the OSA Framework prior to offering and activating applications.

The Framework API contains interfaces between the Application Server and the Framework, and between Network Service Capability Server (SCS) and the Framework (these interfaces are represented by the yellow circles in the figure below). The description of the Framework in the present document separates the interfaces into two distinct sets: Framework to Application interfaces and Framework to Service interfaces.

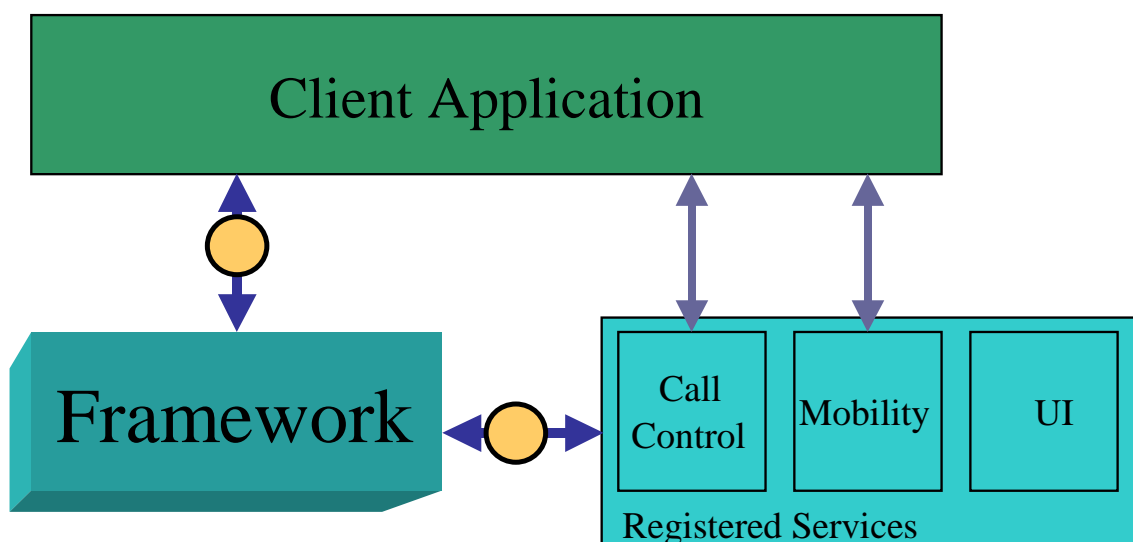


Figure:

Some of the mechanisms are applied only once (e.g. establishment of service agreement), others are applied each time a user subscription is made to an application (e.g. enabling the call attempt event for a new user).

Basic mechanisms between Application and Framework:

- **Authentication:** Once an off-line service agreement exists, the application can access the authentication interface. The authentication model of OSA is a peer-to-peer model, but authentication does not have to be mutual. The application must be authenticated before it is allowed to use any other OSA interface. It is a policy decision for the application whether it must authenticate the framework or not. It is a policy decision for the framework whether it allows an application to authenticate it before it has completed its authentication of the application.

- **Authorisation:** Authorisation is distinguished from authentication in that authorisation is the action of determining what a previously authenticated application is allowed to do. Authentication shall precede authorisation. Once authenticated, an application is authorised to access certain SCFs.
- **Discovery of Framework and network SCFs:** After successful authentication, applications can obtain available Framework interfaces and use the discovery interface to obtain information on authorised network SCFs. The Discovery interface can be used at any time after successful authentication.
- **Establishment of service agreement:** Before any application can interact with a network SCF, a service agreement shall be established. A service agreement may consist of an off-line (e.g. by physically exchanging documents) and an on-line part. The application has to sign the on-line part of the service agreement before it is allowed to access any network SCF.
- **Access to network SCFs:** The Framework shall provide access control functions to authorise the access to SCFs or service data for any API method from an application, with the specified security level, context, domain, etc.

Basic mechanism between Framework and Service Capability Server (SCS):

- **Registering of network SCFs:** SCFs offered by a SCS can be registered at the Framework. In this way the Framework can inform the Applications upon request about available SCFs (Discovery). For example, this mechanism is applied when installing or upgrading an SCS.

The following clauses describe each aspect of the Framework in the following order:

- The *sequence diagrams* give the reader a practical idea of how the Framework is implemented.
- The *class diagrams* clause shows how each of the interfaces applicable to the Framework relate to one another.
- The *interface specification* clause describes in detail each of the interfaces shown within the class diagram part.
- The *State Transition Diagrams (STD)* show the transition between states in the Framework. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions.
- The *data definitions* clause shows a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the common data types part of the present document (29.198-2).

An implementation of this API which supports or implements a method described in the present document, shall support or implement the functionality described for that method, for at least one valid set of values for the parameters of that method. Where a method is not supported by an implementation of a Framework or Service interface, the exception P_METHOD_NOT_SUPPORTED shall be returned to any call of that method. Where a method is not supported by an implementation of an Application interface, a call to that method shall be possible, and no exception shall be returned.

5 The Base Interface Specification

5.1 Interface Specification Format

This clause defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

5.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for client applications are denoted by classes with name Ip<name>. The callback interfaces to the applications are denoted by classes with name IpApp<name>. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>.

5.1.2 Method descriptions

Each method (API method 'call') is described. Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a 'Req' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a 'Res' or 'Err' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name>` or `IpSvc<name>` interfaces to provide the callback mechanism.

5.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

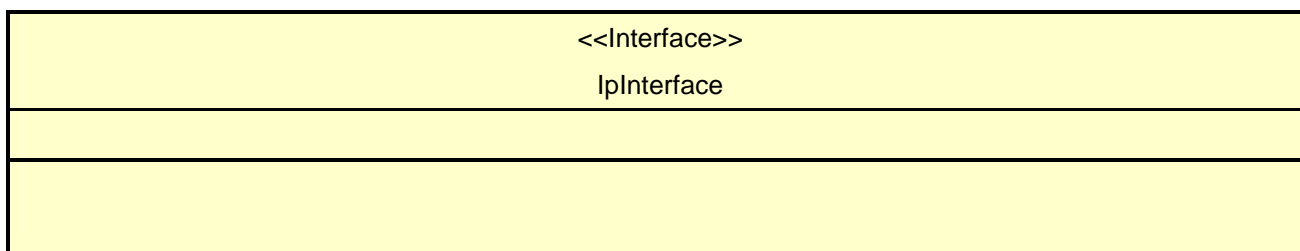
5.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

5.2 Base Interface

5.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.



5.3 Service Interfaces

5.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

5.4 Generic Service Interface

5.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.

<<Interface>> IpService
setCallback (appInterface : in IpInterfaceRef) : void setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : void

5.4.1.1 Method setCallback()

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application. It is not allowed to invoke this method on an interface that uses SessionIDs. Multiple invocations of this method on an interface shall result in multiple callback references being specified. The SCS shall use the most recent callback interface provided by the application using this method. In the event that a callback reference fails or is no longer available, the next most recent callback reference available shall be used.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks.

Raises

TpCommonExceptions, P_INVALID_INTERFACE_TYPE

5.4.1.2 Method setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg. It is not allowed to invoke this method on an interface that does not use SessionIDs. Multiple invocations of this method on an interface shall result in multiple callback references being specified. The SCS shall use the most recent callback interface provided by the application using this method. In the event that a callback reference fails or is no longer available, the next most recent callback reference available shall be used.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks.

sessionID : in TpSessionID

Specifies the session for which the service can invoke the application's callback interface.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_INTERFACE_TYPE

6 Framework Access Session API

6.1 Sequence Diagrams

6.1.1 Trust and Security Management Sequence Diagrams

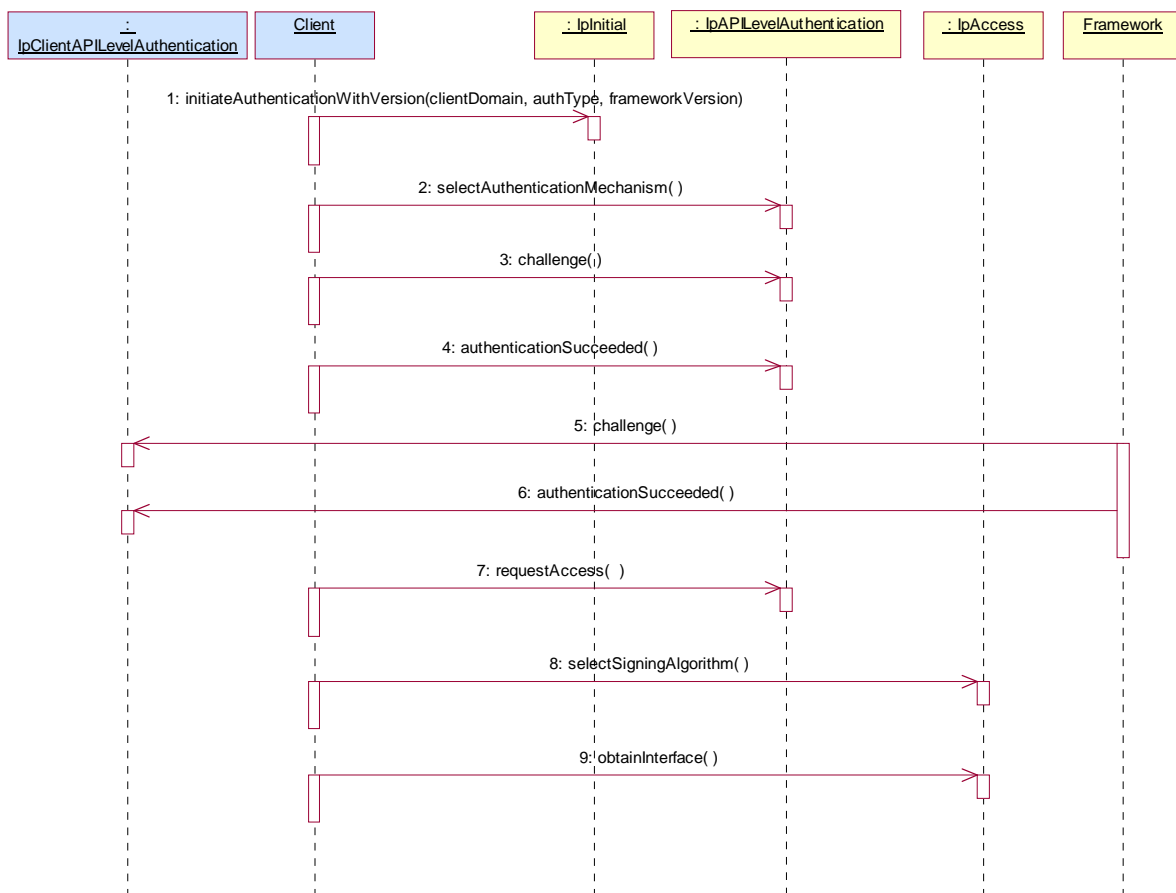
6.1.1.1 Initial Access

The following figure shows a client accessing the OSA Framework for the first time.

Before being authorized to use the OSA SCFs, the client must first of all authenticate itself with the Framework. For this purpose the client needs a reference to the Initial Contact interfaces for the Framework; this may be obtained through a URL, a Naming or Trading Service or an equivalent service, a stringified object reference, etc. At this stage, the client has no guarantee that this is a Framework interface reference, but it is to initiate the authentication process with the Framework. The Initial Contact interface supports the initiateAuthenticationWithVersion method to allow the authentication process to take place.

Once the client has been authenticated by the Framework, it can gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method, by which the client requests a certain type of access SCF.

Independently, the client could decide to authenticate the Framework, before deciding to continue using the interfaces provided by the Framework.



1: Initiate Authentication

The client invokes `initiateAuthenticationWithVersion` on the Framework's "public" (initial contact) interface to initiate the authentication process. It provides in turn a reference to its own authentication interface. The Framework returns a reference to its authentication interface.

2: Select Authentication Mechanism

The client invokes `selectAuthenticationMechanism` on the Framework's API Level Authentication interface, identifying the authentication algorithm it supports for use with CHAP authentication. The Framework prescribes the method to be used. OSA authentication is based on CHAP, which prescribes the MD5 hashing algorithm as the minimum to be supported. Note however that the framework need not accept this algorithm.

3: The client authenticates the Framework, issuing a challenge in the `challenge()` method.

4: The client provides an indication if authentication succeeded.

5: The Framework authenticates the client. The sequence diagram illustrates one of a series of one or more invocations of the challenge method on the client's API Level Authentication interface. In each invocation, the Framework supplies a challenge and the client returns the correct response. The Framework could authenticate the client before the client authenticates the Framework, or afterwards, or the two authentication processes could be interleaved. However, the client shall respond immediately to any challenge issued by the Framework, as the Framework might not respond to any challenge issued by the client until the Framework has successfully authenticated the client.

6: The Framework provides an indication if authentication succeeded.

7: Request Access

Upon successful authentication of the client by the Framework, the client is permitted to invoke `requestAccess` on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to a framework Access interface that is unique for this client. The success or failure of the client's authentication of the Framework does not affect the client's right to invoke `requestAccess`.

8: The client and framework negotiate the signing algorithm to be used for any signed exchanges.

9: The client invokes `obtainInterface` or `obtainInterfaceWithCallback` on the framework's Access interface. This is used to obtain a reference to a framework interface that supports the required framework functionality, such as service discovery, integrity management, service subscription etc.

6.1.1.2 Framework Terminates Access

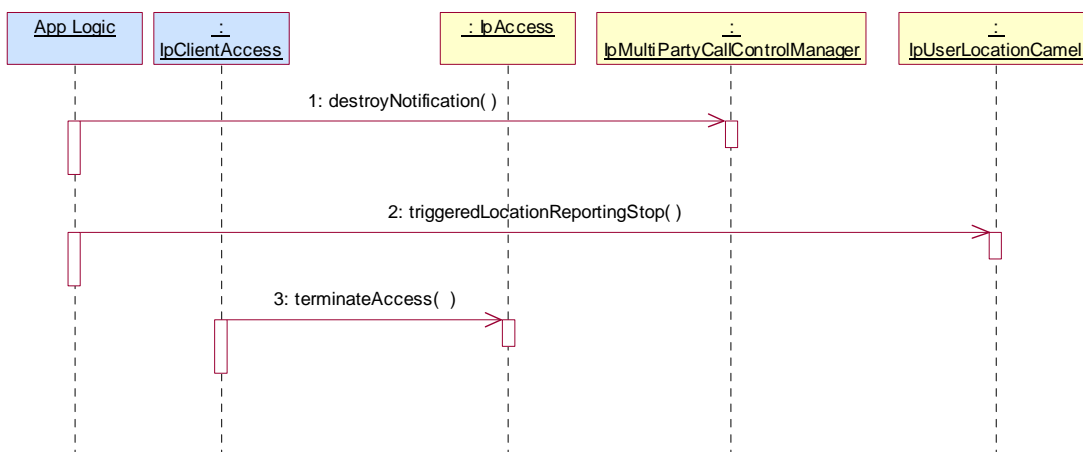
This sequence shows how a Framework could terminate an application's use of the Framework and of all service instances. This type of termination is unusual, but possible with the `terminateAccess` method. Note that if at any point the framework's level of confidence in the identity of the client becomes too low, perhaps due to re-authentication failing, the framework should terminate all outstanding service agreements for that client, and should take steps to terminate the client's access session **WITHOUT** invoking `terminateAccess()` on the client. This follows a generally accepted security model where the framework has decided that it can no longer trust the client and will therefore sever ALL contact with it.



- 1: Following successful authentication and service discovery, the client initiates the service agreement signing process (not shown). This is completed when the client invokes signServiceAgreement on the Framework's IpServiceAgreementManagement interface, and a reference to an instance of a service manager interface is returned.
- 2: The client (application) had initiated service agreement signing process for a second service agreement (not shown), and when the client signs this second service agreement, a reference to an instance of another service manager, for another service type, is returned.
- 3: The application starts to use the new service manager interface.
- 4: The application starts to use the other new service manager interface.
- 5: The framework decides to terminate the application's access session, and to terminate all its service agreements. This is an unusual and drastic step, but could be e.g. due to violation or expiry of the application's service agreements, or some problem within the framework itself. The framework will also destroy each of the service managers the application was using (not shown). The application is now no longer authenticated with the framework, and all Framework and service interfaces it was using are destroyed.

6.1.1.3 Application Terminates Access

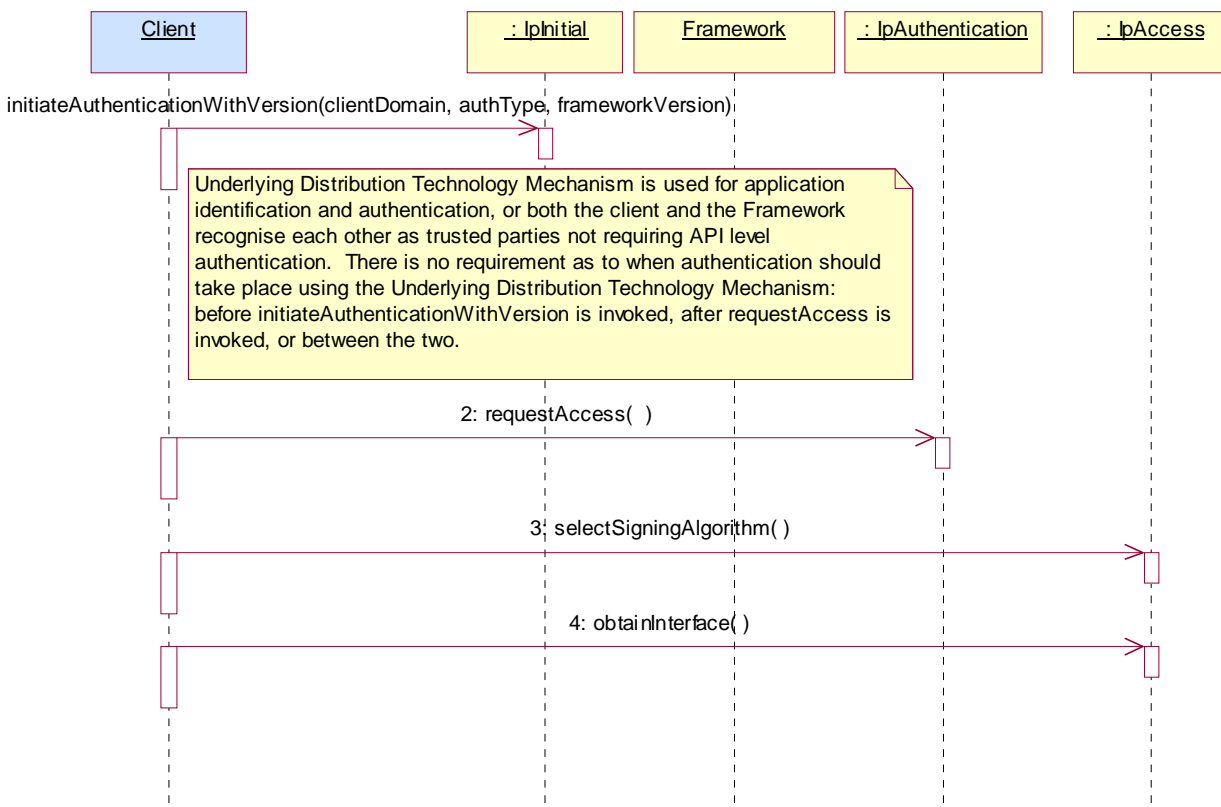
This sequence shows how an application could terminate its use of the Framework and of all service instances. This type of termination is unusual, but possible with the terminateAccess method.



- 1: The application terminates its use of the multi-party call control service manager in a controlled manner.
- 2: The application ceases to use the user location camel SCF.
- 3: The application decides to terminate its access session and all its service agreements in one go. The framework will also destroy each of the service managers the application was using (not shown). The application is now no longer authenticated with the framework, and all Framework and service interfaces it was using are destroyed. The application could have terminated its service agreements one by one, by invoking `terminateServiceAgreement` on the Framework's `IpServiceAgreementManager` interface, and then invoked `terminateAccess` on the Framework's `IpAccess` interface, which would have been a more controlled shutdown.

6.1.1.4 Non-API level Authentication

The following figure shows a client accessing the OSA Framework for the first time. The client and the framework have mutually authenticated one another using an underlying distribution technology mechanism, or the client and the framework recognise each other as a trusted party, not requiring authentication.



- 1: The client calls `initiateAuthenticationWithVersion` on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. In this case, the client selects to use the underlying distribution technology mechanism for identification and authentication. What that mechanism is, if it even exists, is outside the scope of the API.
- 2: The client invokes the `requestAccess` method on the Framework's Authentication interface. This returns a reference to the framework Access interface that is unique for the client.
- 3: If the authentication was successful, the client and the framework can negotiate, on the framework's Access interface, the signing algorithm to be used for any signed exchanges.

4: The client can now invoke `obtainInterface` or `obtainInterfaceWithCallback` on the framework's `Access` interface. This is used to obtain a reference to a framework interface such as service discovery, integrity management, service subscription etc.

6.1.1.5 API Level Authentication

This sequence diagram illustrates the two-way mechanism by which the client and the framework mutually authenticate one another.

The OSA API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. The inclusion of cryptographic processes and digital signatures in the authentication procedure depends on the type of authentication technique selected. In some cases strong authentication may need to be enforced by the Framework to prevent misuse of resources. In addition it may be necessary to define the minimum encryption key length that can be used to ensure a high degree of confidentiality.

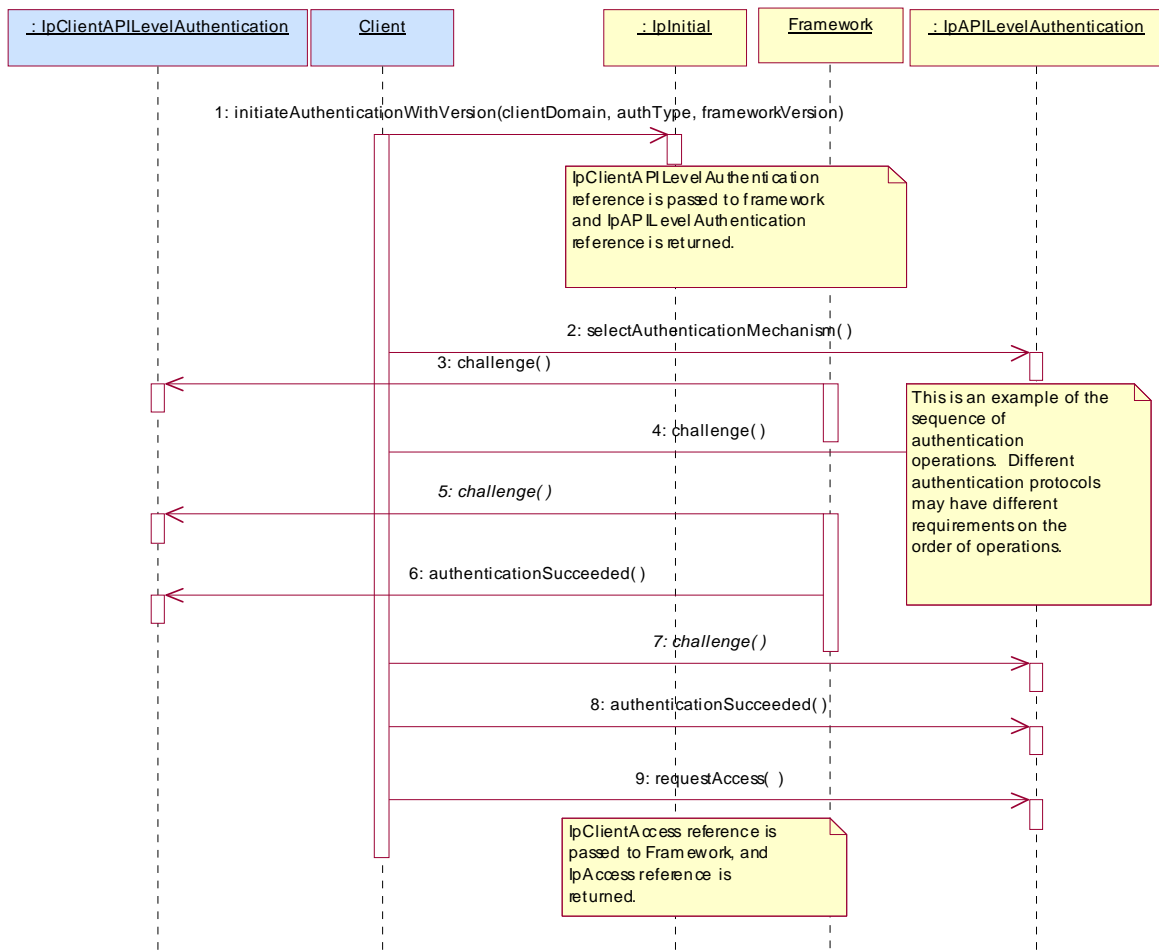
The client must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

1) The client calls `initiateAuthenticationWithVersion` on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. This authentication process may be specific to the provider, or the implementation technology used. The `initiateAuthenticationWithVersion` method can be used to specify the specific process, (e.g. CORBA security). OSA defines a generic authentication interface (API Level Authentication), which can be used to perform the authentication process. The `initiateAuthenticationWithVersion` method allows the client to pass a reference to its own authentication interface to the Framework, and receive a reference to the authentication interface preferred by the client, in return. In this case the API Level Authentication interface.

2) The client invokes the `selectAuthenticationMechanism` on the Framework's API Level Authentication interface. This includes the authentication algorithms supported by the client. The framework then chooses a mechanism based on the capabilities of the client and the Framework. If the client is capable of handling more than one mechanism, then the Framework chooses one option, defined in the `prescribedMethod` parameter. In some instances, the authentication mechanism of the client may not fulfil the demands of the Framework, in which case, the authentication will fail, for example CHAP prescribes the MD5 hashing algorithm as the minimum to be supported, however the framework need not accept this algorithm.

3) The application and Framework interact to authenticate each other by using the challenge method. For an authentication method of `P_OSA_AUTHENTICATION`, this procedure consists of a number of challenge/ response exchanges. This authentication protocol is performed using the challenge method on the API Level Authentication interface. `P_OSA_AUTHENTICATION` is based on CHAP, which is primarily a one-way protocol. There are in fact two authentication processes: authentication of the client performed by the Framework, and authentication of the Framework performed by the client. Mutual authentication is achieved by both these processes terminating successfully. Mutual authentication may not necessarily be required, i.e. it could be that a client may not need to authenticate the Framework. There is also no required order for the execution of these two authentication processes, however, the client shall respond immediately to any challenge issued by the Framework, as the Framework might not respond to any challenge issued by the client until the Framework has successfully authenticated the client.

Note that at any point during the access session, either side can request re-authentication of the other side.



6.2 Class Diagrams

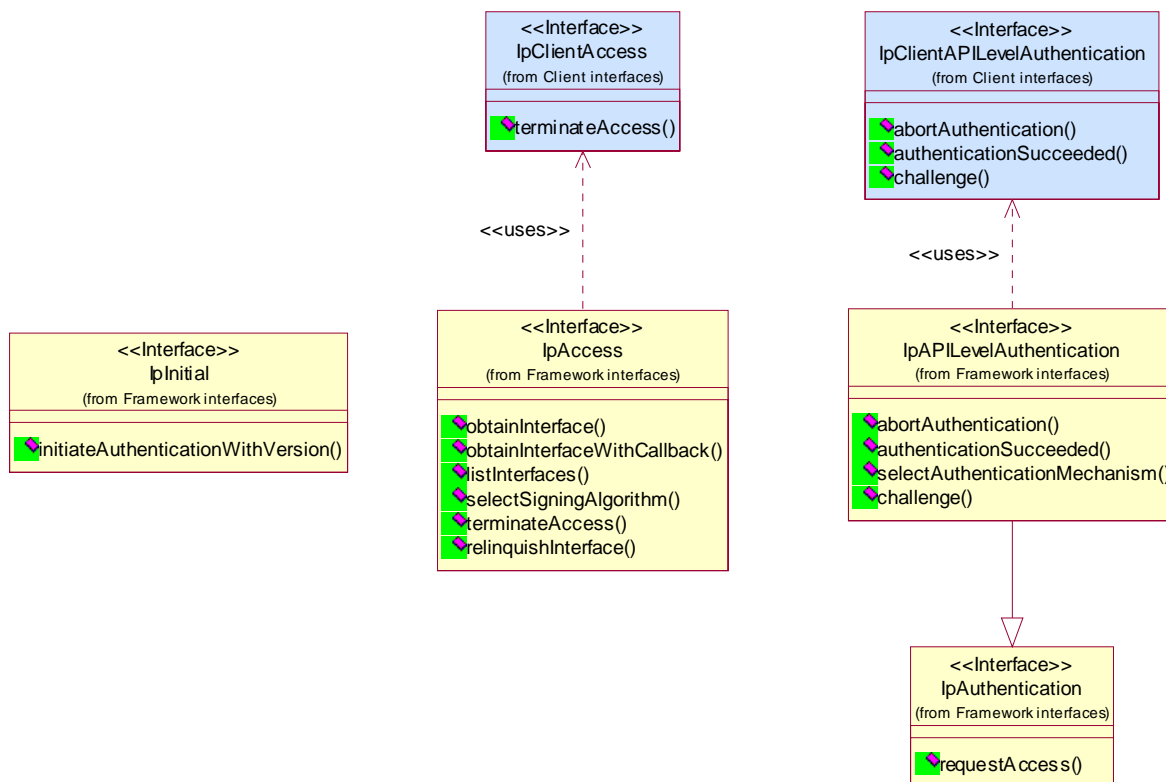


Figure: Trust and Security Management Package Overview

6.3 Interface Classes

6.3.1 Trust and Security Management Interface Classes

The Trust and Security Management Interfaces provide:

- the first point of contact for a client to access a Framework provider;
- the authentication methods for the client and Framework provider to perform an authentication protocol;
- the client with the ability to select a service capability feature to make use of;
- the client with a portal to access other Framework interfaces.

The process by which the client accesses the Framework provider has been separated into 3 stages, each supported by a different Framework interface:

- 1) Initial Contact with the Framework;
- 2) Authentication;
- 3) Access to Framework and Service Capability Features.

6.3.1.1 Interface Class IpClientAPILevelAuthentication

Inherits from: IpInterface.

If the IpClientAPILevelAuthentication interface is implemented by a client, `challenge()`, `abortAuthentication()` and `authenticationSucceeded()` methods shall be implemented.

<<Interface>> IpClientAPILevelAuthentication
abortAuthentication () : void authenticationSucceeded () : void challenge (challenge : in TpOctetSet) : TpOctetSet

6.3.1.1.1 Method abortAuthentication()

The framework uses this method to abort the authentication process where the client is authenticating the Framework. This method is invoked if the framework wishes to abort the authentication process before it has been authenticated by the client, (unless the client responded incorrectly to a challenge in which case no further communication with the client should occur.) Calls to this method after the Framework has been authenticated by the client shall not result in an immediate removal of the Framework's authentication (the client may wish to authenticate the Framework again, however).

Parameters

No Parameters were identified for this method.

6.3.1.1.2 Method authenticationSucceeded()

The Framework uses this method to inform the client of the success of the authentication attempt. The client may invoke requestAccess on the Framework's APILevelAuthentication interface following invocation of this method.

Parameters

No Parameters were identified for this method.

6.3.1.1.3 Method challenge()

This method is used by the framework to authenticate the client. The client must respond with the correct responses to the challenges presented by the framework. The number of exchanges is dependent on the policies of each side. The authentication of the client is deemed successful when the authenticationSucceeded method is invoked by the Framework.

The invocation of this method may be interleaved with challenge() calls by the client on the IpAPILevelAuthentication interface. The client shall respond immediately to authentication challenges from the Framework, and not wait until the Framework has responded to any challenge the client may issue.

Returns <response> : This is the response of the client application to the challenge of the framework in the current sequence. The formatting and construction of this parameter shall be according to section 4.1 of RFC 1994. A complete CHAP Response packet shall be used to carry the response octet set. That octet set will be the result of applying the designated hashing algorithm, which is indicated via the client's invocation of selectAuthenticationMechanism(), to an octet set consisting of the concatenation of the CHAP Identifier, the shared "secret", and the supplied challenge value. The Name field of the CHAP Response packet must be present and contain a valid value in order for the CHAP Response to be valid. However, the Name field is not used in the authentication process.

Steps for constructing the response octet set:

1. Extract the Identifier and Value fields from the CHAP Challenge packet passed in the challenge() method's challenge parameter

2. Build an octet set consisting of the concatenation of the Identifier, the "shared secret", and the Value from the CHAP Challenge
3. Compute the hash of the octet set resulting from the previous step using the designated hashing algorithm
4. Construct a complete CHAP Response packet with the resulting octet set from previous step as the CHAP Value

Steps for validating the response octet set:

1. Verify that the Identifier sent in the original CHAP Challenge matches the Identifier received in the CHAP Response. If it does not, authentication fails.
2. Build an octet set consisting of the concatenation of the original Identifier, the "shared secret", and the original challenge value
3. Compute the hash of the resulting octet set from the previous step using the designated hashing algorithm
4. Verify the octet set resulting from the previous step matches the octet set contained in the Value field of the CHAP Response. A match indicates successful authentication.

Parameters

challenge : in TpOctetSet

The challenge presented by the framework to be responded to by the client. The challenge format used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol (RFC 1994).

The challenge octet set must be formatted as a CHAP Challenge packet as defined in section 4.1 of RFC 1994. A complete and properly formatted CHAP Challenge packet must be used. The Name field of the CHAP Challenge packet must be present and contain a valid value in order for the CHAP Response to be valid. However, the Name field is not used in the authentication process.

Steps for constructing the challenge octet set:

1. Create a random challenge value (octet set). Per RFC 1994, this value must be between 1 and 255 octets in length.
2. Construct a CHAP Challenge packet based on 4.1 of RFC 1994 with the octet set from the previous step passed in the Value field within the CHAP Challenge.

Returns

TpOctetSet

6.3.1.2 Interface Class IpClientAccess

Inherits from: IpInterface.

IpClientAccess interface is offered by the client to the framework to allow it to initiate interactions during the access session. This interface and the terminateAccess() method shall be implemented by a client.

<<Interface>> IpClientAccess
terminateAccess (terminationText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : in TpOctetSet) : void

6.3.1.2.1 Method terminateAccess()

The terminateAccess operation is used by the framework to end the client's access session.

After terminateAccess() is invoked, the client will no longer be authenticated with the framework. The client will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail. The framework shall also identify and terminate all remaining service instances that apply as a result of the client access termination. If at any point the framework's level of confidence in the identity of the client becomes too low, perhaps due to re-authentication failing, the framework should terminate all outstanding service agreements for that client, and should take steps to terminate the client's access session WITHOUT invoking terminateAccess() on the client. This follows a generally accepted security model where the framework has decided that it can no longer trust the client and will therefore sever ALL contact with it.

Parameters

terminationText : in TpString

This is the termination text describes the reason for the termination of the access session.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. It shall be identical to the one chosen by the framework in response to IpAccess.selectSigningAlgorithm(). If the signingAlgorithm is not the chosen one, is invalid, or unknown to the client, the P_INVALID_SIGNING_ALGORITHM exception will be thrown. The list of possible algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in RFC 2630) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the termination text. The "external signature" construct shall not be used (i.e. the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The framework uses this to confirm its identity to the client. The client can check that the terminationText has been signed by the framework. If a match is made, the access session is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNING_ALGORITHM, P_INVALID_SIGNATURE

6.3.1.3 Interface Class IpInitial

Inherits from: IpInterface.

The Initial Framework interface is used by the client to initiate the authentication with the Framework. This interface and the initiateAuthenticationWithVersion() method shall be implemented by a Framework.

<<Interface>> IpInitial
initiateAuthenticationWithVersion (clientDomain : in TpAuthDomain, authType : in TpAuthType, frameworkVersion : in TpVersion) : TpAuthDomain

6.3.1.3.1 Method initiateAuthenticationWithVersion()

This method is invoked by the client to start the process of authentication with the framework, and request the use of a specific authentication method using the new method with support for backward compatibility in the framework. The returned fwDomain authInterface will be selected to match the proposed version from the Client in the Framework response. If the Framework cannot work with the proposed framework version the framework returns an error code (P_INVALID_VERSION).

Returns <fwDomain> : This provides the client with a framework identifier, and a reference to call the authentication interface of the framework.

```

structure TpAuthDomain {
    domainID:    TpDomainID;
    authInterface: IpInterfaceRef;
};

```

The domainID parameter is an identifier for the framework (i.e. TpFwID). It is used to identify the framework to the client.

The authInterface parameter is a reference to the authentication interface of the framework that is unique for each requesting client. The type of this interface is defined by the authType parameter. The client uses this interface to authenticate with the framework.

Note, there are no identifiers used in the authentication interface to correlate requests and responses, therefore the authentication interface may not be shared amongst multiple clients.

Parameters

clientDomain : in TpAuthDomain

This identifies the client domain to the framework, and provides a reference to the authentication interface.

```

structure TpAuthDomain {
    domainID:    TpDomainID;
    authInterface: IpInterfaceRef;
};

```

The domainID parameter is an identifier either for a client application (i.e. TpClientAppID) or for an enterprise operator (i.e. TpEntOpID), or for an instance of a service for which a client application has signed a service agreement (i.e. TpServiceInstanceID), or for a service supplier (i.e. TpServiceSupplierID). It is used to identify the client domain to the framework, (see challenge() on IpAPILevelAuthentication). If the framework does not recognise the domainID, the framework returns an error code (P_INVALID_DOMAIN_ID).

A client application (identifiable by a given TpClientAppID) may optionally initiate authentication with the Framework by invoking this method multiple times. The Framework may elect to reject these subsequent requests, or may choose to associate them together as independent sessions under the same TpClientAppID.

The authInterface parameter is a reference to call the authentication interface of the client. The type of this interface is defined by the authType parameter. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

authType : in TpAuthType

This identifies the type of authentication mechanism requested by the client. It provides operators and clients with the opportunity to use an alternative to the API level Authentication interface, e.g. an implementation specific authentication mechanism like CORBA Security, using the IpAuthentication interface, or Operator specific Authentication interfaces. OSA API level Authentication is the default authentication mechanism (P_OSA_AUTHENTICATION). If P_OSA_AUTHENTICATION is selected, then the clientDomain and fwDomain authInterface parameters are references to interfaces of type Ip(Client)APILevelAuthentication. If P_AUTHENTICATION is selected, the fwDomain authInterface parameter references to interfaces of type IpAuthentication that is used when an underlying distribution technology authentication mechanism is used.

frameworkVersion : in TpVersion

This identifies the version of the Framework implemented in the client. The TpVersion is a String containing the version number. Valid version numbers are defined in the respective framework specification.

Returns

TpAuthDomain

Raises

TpCommonExceptions, P_INVALID_DOMAIN_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_AUTH_TYPE, P_INVALID_VERSION

6.3.1.4 Interface Class IpAuthentication

Inherits from: IpInterface.

The Authentication Framework interface is used by client to request access to other interfaces supported by the Framework. The authentication process should in this case be done with some underlying distribution technology authentication mechanism, e.g. CORBA Security.

At least one of IpAuthentication or IpAPILevelAuthentication interfaces shall be implemented by a Framework as a minimum requirement. The requestAccess() method shall be implemented in each.

<<Interface>> IpAuthentication
requestAccess (accessType : in TpAccessType, clientAccessInterface : in IpInterfaceRef) : IpInterfaceRef

6.3.1.4.1 Method requestAccess()

Once the client has been authenticated by the framework, the client may invoke the requestAccess operation on the IpAuthentication or IpAPILevelAuthentication interface. This allows the client to request the type of access they require. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Operators can define their own access interfaces to satisfy client requirements for different types of access.)

If this method is called before the client has been successfully authenticated, then the request fails, and an error code (P_ACCESS_DENIED) is returned.

This method may be invoked by the client immediately on IpAuthentication, when API Level authentication is not being used, since there is no indication to the client at API level that it is authenticated with the Framework.

Returns <fwAccessInterface> : This provides the reference for the client to call the access interface of the framework. The access reference provided is unique to the requesting client.

Parameters

accessType : in TpAccessType

This identifies the type of access interface requested by the client. If the framework does not provide the type of access identified by accessType, then an error code (P_INVALID_ACCESS_TYPE) is returned.

clientAccessInterface : in IpInterfaceRef

This provides the reference for the framework to call the access interface of the client. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

Returns

IpInterfaceRef

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_ACCESS_TYPE, P_INVALID_INTERFACE_TYPE

6.3.1.5 Interface Class IpAPILevelAuthentication

Inherits from: IpAuthentication.

The API Level Authentication Framework interface is used by the client to authenticate the Framework. It is also used to initiate the authentication process.

If the IpAPILevelAuthentication interface is implemented by a Framework, then selectAuthenticationMechanism(), challenge(), abortAuthentication() and authenticationSucceeded () shall be implemented. IpAPILevelAuthentication inherits the requirements of IpAuthentication, therefore requestAccess() shall be implemented.

<<Interface>> IpAPILevelAuthentication
abortAuthentication () : void authenticationSucceeded () : void selectAuthenticationMechanism (authMechanismList : in TpAuthMechanismList) : TpAuthMechanism challenge (challenge : in TpOctetSet) : TpOctetSet

6.3.1.5.1 Method abortAuthentication()

The client uses this method to abort the authentication process where the framework is authenticating the client. This method is invoked if the client no longer wishes to continue the authentication process, (unless the framework responded incorrectly to a challenge in which case no further communication with the framework should occur.) If this method has been invoked before the client has been authenticated by the Framework, calls to the requestAccess operation on IpAPILevelAuthentication will return an error code (P_ACCESS_DENIED), until the client has been properly authenticated. If this method is invoked after the client has been authenticated by the Framework, it shall not result in the immediate removal of the client's authentication. (The Framework may wish to authenticate the client again, however).

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions, P_ACCESS_DENIED

6.3.1.5.2 Method authenticationSucceeded()

The client uses this method to inform the framework of the success of the authentication attempt. Calls to this method have no impact on the client's rights to call requestAccess(), which depend exclusively on the framework's successful authentication of the client.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions, P_ACCESS_DENIED

6.3.1.5.3 Method selectAuthenticationMechanism()

The client uses this method to inform the Framework of the different authentication mechanisms it supports as part of API level Authentication. The Framework will select one of the suggested authentication mechanisms and that mechanism shall be used for authentication by both Framework and Client. The authentication mechanism chosen as a result of the response to this method remains valid for an instance of IpAPILevelAuthentication and until this method is re-invoked by the client. If a mechanism that is acceptable to the framework within the capability of the client cannot be found, the framework throws the P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM exception.

Returns: selectedMechanism. This is the authentication mechanism chosen by the Framework. The chosen mechanism shall be taken from the list of mechanisms proposed by the Client.

Parameters

authMechanismList : in TpAuthMechanismList

The list of authentication mechanisms supported by the client.

Returns

TpAuthMechanism

Raises

**TpCommonExceptions, P_ACCESS_DENIED,
P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM**

6.3.1.5.4 Method challenge()

This method is used by the client to authenticate the framework. The framework must respond with the correct responses to the challenges presented by the client. The domainID received in the initiateAuthenticationWithVersion() can be used by the framework to reference the correct public key for the client (the key management system is currently outside of the scope of the OSA APIs). The number of exchanges is dependent on the policies of each side. The authentication of the framework is deemed successful when the authenticationSucceeded method is invoked by the client.

The invocation of this method may be interleaved with challenge() calls by the framework on the client's APILevelAuthentication interface.

Returns <response> : This is the response of the framework to the challenge of the client in the current sequence. The formatting and construction of this parameter shall be according to section 4.1 of RFC 1994. A complete CHAP Response packet shall be used to carry the response octet set. That octet set will be the result of applying the designated hashing algorithm, which is indicated via the client's invocation of selectAuthenticationMechanism(), to an octet set consisting of the concatenation of the CHAP Identifier, the shared "secret", and the supplied challenge value. The Name field of the CHAP Response packet must be present and contain a valid value in order for the CHAP Response to be valid. However, the Name field is not used in the authentication process.

Steps for constructing the response octet set:

1. Extract the Identifier and Value fields from the CHAP Challenge packet passed in the challenge() method's challenge parameter
2. Build an octet set consisting of the concatenation of the Identifier, the "shared secret", and the Value from the CHAP Challenge
3. Compute the hash of the octet set resulting from the previous step using the designated hashing algorithm
4. Construct a complete CHAP Response packet with the resulting octet set from previous step as the CHAP Value

Steps for validating the response octet set:

1. Verify that the Identifier sent in the original CHAP Challenge matches the Identifier received in the CHAP Response. If it does not, authentication fails.
2. Build an octet set consisting of the concatenation of the original Identifier, the "shared secret", and the original challenge value
3. Compute the hash of the resulting octet set from the previous step using the designated hashing algorithm
4. Verify the octet set resulting from the previous step matches the octet set contained in the Value field of the CHAP Response. A match indicates successful authentication.

Parameters

challenge : in TpOctetSet

The challenge presented by the client to be responded to by the framework. The challenge format used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol (RFC 1994).

The challenge octet set must be formatted as a CHAP Challenge packet as defined in section 4.1 of RFC 1994. A complete and properly formatted CHAP Challenge packet must be used. The Name field of the CHAP Challenge packet must be present and contain a valid value in order for the CHAP Response to be valid. However, the Name field is not used in the authentication process.

Steps for constructing the challenge octet set:

1. Create a random challenge value (octet set). Per RFC 1994, this value must be between 1 and 255 octets in length.
2. Construct a CHAP Challenge packet based on 4.1 of RFC 1994 with the octet set from the previous step passed in the Value field within the CHAP Challenge.

Returns

TpOctetSet

Raises

TpCommonExceptions, P_ACCESS_DENIED

6.3.1.6 Interface Class IpAccess

Inherits from: IpInterface.

This interface shall be implemented by a Framework. As a minimum requirement the obtainInterface() and obtainInterfaceWithCallback(), selectSigningAlgorithm() and terminateAccess() methods shall be implemented.

<<Interface>> IpAccess
<pre> obtainInterface (interfaceName : in TpInterfaceName) : IpInterfaceRef obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, clientInterface : in IpInterfaceRef) : IpInterfaceRef listInterfaces () : TpInterfaceNameList selectSigningAlgorithm (signingAlgorithmCaps : in TpSigningAlgorithmCapabilityList) : TpSigningAlgorithm terminateAccess (terminationText : in TpString, digitalSignature : in TpOctetSet) : void relinquishInterface (interfaceName : in TpInterfaceName, terminationText : in TpString, digitalSignature : in TpOctetSet) : void </pre>

6.3.1.6.1 Method obtainInterface()

This method is used to obtain other framework interfaces. The client uses this method to obtain interface references to other framework interfaces. (The obtainInterfaceWithCallback method should be used if the client is required to supply a callback interface to the framework.)

Returns <fwInterface> : This is the reference to the interface requested.

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

Returns

IpInterfaceRef

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME

6.3.1.6.2 Method obtainInterfaceWithCallback()

This method is used to obtain other framework interfaces. The client uses this method to obtain interface references to other framework interfaces, when it is required to supply a callback interface to the framework. (The obtainInterface method should be used when no callback interface needs to be supplied.)

Returns <fwInterface> : This is the reference to the interface requested.

*Parameters***interfaceName : in TpInterfaceName**

The name of the framework interface to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

clientInterface : in IpInterfaceRef

This is the reference to the client interface, which is used for callbacks. If a client interface is not needed, then this method should not be used. (The obtainInterface method should be used when no callback interface needs to be supplied.) If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

*Returns***IpInterfaceRef***Raises***TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME, P_INVALID_INTERFACE_TYPE****6.3.1.6.3 Method listInterfaces()**

The client uses this method to obtain the names of all interfaces supported by the framework. It can then obtain the interfaces it wishes to use using either obtainInterface() or obtainInterfaceWithCallback().

Returns <frameworkInterfaces> : The frameworkInterfaces parameter contains a list of interfaces that the framework makes available.

Parameters

No Parameters were identified for this method.

*Returns***TpInterfaceNameList***Raises***TpCommonExceptions, P_ACCESS_DENIED****6.3.1.6.4 Method selectSigningAlgorithm()**

The client uses this method to inform the Framework of the different signing algorithms it supports for use in all cases where digital signatures are required. The Framework will select one of the suggested algorithms. This method shall be the first method invoked by the client on IpAccess. The algorithm chosen as a result of the response to this method remains valid for an instance of IpAccess and until this method is re-invoked by the client.

Subsequent invocations of selectSigningAlgorithm() may change the signing algorithm used during the access session. However, once signServiceAgreement() has been called on the client by the framework, the signing algorithm currently selected must be used for the client's invocation of signServiceAgreement() on the Framework as well as for subsequent calls to terminateServiceAgreement(). Other operations requiring digital signatures will use the latest algorithm specified by selectSigningAlgorithm().

If an algorithm that is acceptable to the framework within the capability of the client cannot be found, the framework throws the P_NO_ACCEPTABLE_SIGNING_ALGORITHM exception.

Returns: selectedAlgorithm. This is the signing algorithm chosen by the Framework. The chosen algorithm shall be taken from the list proposed by the Client.

Parameters

signingAlgorithmCaps : in TpSigningAlgorithmCapabilityList

The list of signing algorithms supported by the client.

Returns

TpSigningAlgorithm

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_NO_ACCEPTABLE_SIGNING_ALGORITHM

6.3.1.6.5 Method terminateAccess()

The terminateAccess method is used by the client to request that its access session with the framework is ended. After it is invoked, the client will no longer be authenticated with the framework. The client will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail. Also, all remaining service instances created by the framework either directly in this access session or on behalf of the client during this access session shall be terminated.

Parameters

terminationText : in TpString

This is the termination text describes the reason for the termination of the access session.

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in RFC 2630) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630 using the latest signing algorithm selected with selectSigningAlgorithm(). The content is made of the termination text. The "external signature" construct shall not be used (i.e. the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The client uses this to confirm its identity to the framework. The framework can check that the terminationText has been signed by the client. If a match is made, the access session is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNATURE

6.3.1.6.6 Method relinquishInterface()

The client uses this method to release an instance of a framework interface that was obtained during this access session.

Parameters

interfaceName : in TpInterfaceName

This is the name of the framework interface which is being released. If the interfaceName is invalid, the framework throws the P_INVALID_INTERFACE_NAME exception. If the interface has not been given to the client during this access session, then the P_TASK_REFUSED exception will be thrown.

terminationText : in TpString

This is the termination text describes the reason for the release of the interface. This text is required simply because the digitalSignature parameter requires a terminationText to sign.

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in RFC 2630) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630 using the latest signing algorithm selected with `selectSigningAlgorithm()`. The content is made of the termination text. The "external signature" construct shall not be used (i.e. the `eContent` field in the `EncapsulatedContentInfo` field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The client uses this to confirm its identity to the framework. The framework can check that the `terminationText` has been signed by the client. If a match is made, the interface is released, otherwise the `P_INVALID_SIGNATURE` exception will be thrown.

Raises

`TpCommonExceptions`, `P_INVALID_SIGNATURE`, `P_INVALID_INTERFACE_NAME`

6.4 State Transition Diagrams

This clause contains the State Transition Diagrams for the objects that implement the Framework interfaces on the gateway side. The State Transition Diagrams show the behaviour of these objects. For each state the methods that can be invoked by the client are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the client also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

6.4.1 Trust and Security Management State Transition Diagrams

6.4.1.1 State Transition Diagrams for IpInitial



Figure : State Transition Diagram for IpInitial

6.4.1.2 State Transition Diagrams for IpAPILevelAuthentication

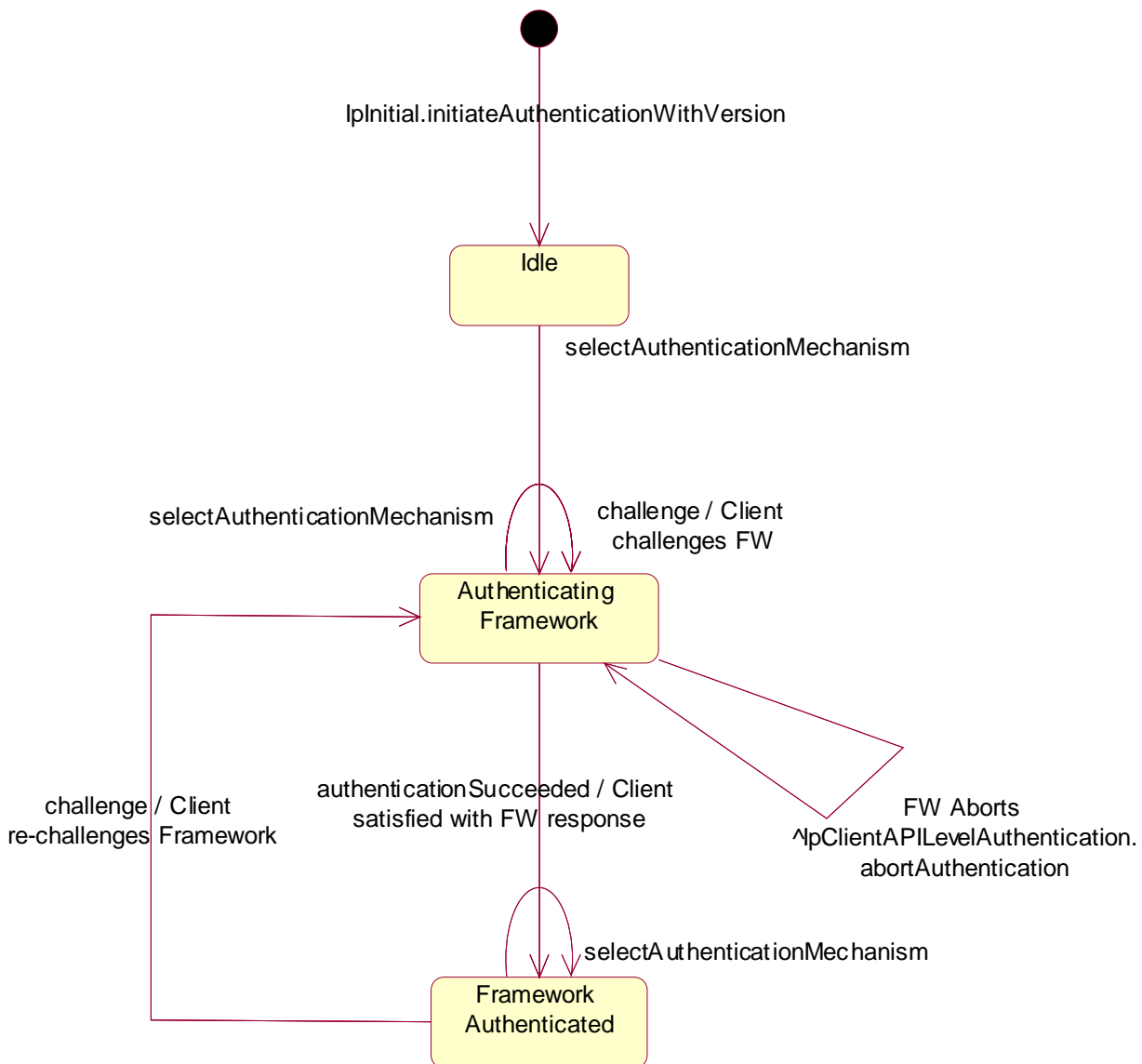


Figure : STD for IpAPILevelAuthentication: Client authenticates Framework using initiateAuthenticationWithVersion() and challenge() method combination

6.4.1.2.1 Idle State

When the client has invoked the IpInitial initiateAuthenticationWithVersion method, an object implementing the IpAPILevelAuthentication interface is created. The client now has to select the authentication mechanism to be used using selectAuthenticationMechanism.

6.4.1.2.2 Authenticating Framework State

When entering this state, the client requests the Framework to authenticate itself. The client invokes the challenge method on the Framework. The Framework may either buffer the requests and respond when the client has been authenticated, or respond immediately, depending on policy. When the client has processed the response from the authenticate request on the Framework, the response is analysed. If the response is valid but the authentication process is not yet complete, then another authenticate request or challenge is sent to the Framework. If the response is valid and the authentication process has been completed, then a transition to the state Framework Authenticated is made and the Framework is informed of its success by invoking authenticationSucceeded. At any time the Framework may abort the

authentication process by calling `abortAuthentication` on the client's `APILevelAuthentication` interface. The client may also call `selectAuthenticationMechanism` to choose another hash algorithm.

6.4.1.2.3 Framework Authenticated State

This state is entered when the client indicates that the Framework has been authenticated, by calling `authenticationSucceeded` on the Framework's `IpAPILevelAuthentication` interface. The client may at any time request re-authentication of the Framework by calling the `challenge` method, resulting in a transition back to `Authenticating Framework` state. The client may also call `selectAuthenticationMechanism` to choose another hash algorithm.

6.4.1.2.4 Authenticating Client State

When entering this state, the Framework requests the client to authenticate itself. The Framework invokes the `challenge` method on the client. When the Framework has processed the response from the `authenticate` request or `challenge` on the client, the response is analysed. If the response is valid but the authentication process is not yet complete, then another `authenticate` request or `challenge` is sent to the client. If the response is valid and the authentication process has been completed, then a transition to the state `Client Authenticated` is made, the client is informed of its success by invoking `authenticationSucceeded`. In case the response is not valid, the `Authentication` object is destroyed. This implies that the client has to re-initiate the authentication by calling once more the `initiateAuthenticationWithVersion` method on the `IpInitial` interface. At any time the client may abort the authentication process by calling `abortAuthentication` on the Framework's `IpAPILevelAuthentication` interface. The client may also call `selectAuthenticationMechanism` to choose another hash algorithm.

6.4.1.2.5 Client Authenticated State

In this state the client is considered authenticated and is now allowed to request access to the `IpAccess` interface. If the framework decides to re-authenticate the client, then the `challenge` is sent to the client and a transition back to the `AuthenticatingClient` state occurs. The client may also call `selectAuthenticationMechanism` to choose another hash algorithm.

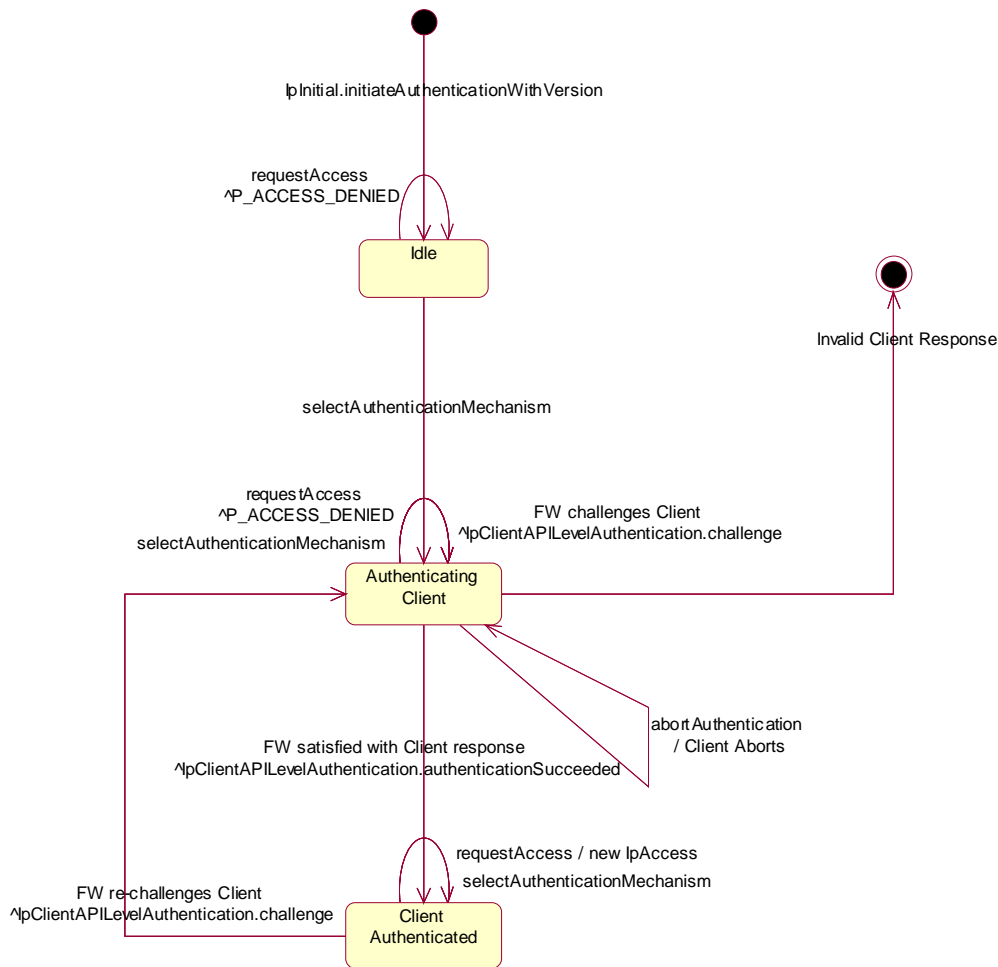


Figure : STD for IpAPILevelAuthentication: Framework authenticates Client using initiateAuthenticationWithVersion() and challenge() method combination

6.4.1.2.6 Idle State

When the client has invoked the IpInitial initiateAuthenticationWithVersion method, an object implementing the IpAPILevelAuthentication interface is created. The client now has to select the authentication mechanism to be used using selectAuthenticationMechanism.

6.4.1.2.7 Authenticating Framework State

When entering this state, the client requests the Framework to authenticate itself. The client invokes the challenge method on the Framework. The Framework may either buffer the requests and respond when the client has been authenticated, or respond immediately, depending on policy. When the client has processed the response from the authenticate request on the Framework, the response is analysed. If the response is valid but the authentication process is not yet complete, then another authenticate request or challenge is sent to the Framework. If the response is valid and the authentication process has been completed, then a transition to the state Framework Authenticated is made and the Framework is informed of its success by invoking authenticationSucceeded. At any time the Framework may abort the authentication process by calling abortAuthentication on the client's APILevelAuthentication interface. The client may also call selectAuthenticationMechanism to choose another hash algorithm.

6.4.1.2.8 Framework Authenticated State

This state is entered when the client indicates that the Framework has been authenticated, by calling authenticationSucceeded on the Framework's IpAPILevelAuthentication interface. The client may at any time request

re-authentication of the Framework by calling the challenge method, resulting in a transition back to Authenticating Framework state. The client may also call selectAuthenticationMechanism to choose another hash algorithm.

6.4.1.2.9 Authenticating Client State

When entering this state, the Framework requests the client to authenticate itself. The Framework invokes the challenge method on the client. When the Framework has processed the response from the authenticate request or challenge on the client, the response is analysed. If the response is valid but the authentication process is not yet complete, then another authenticate request or challenge is sent to the client. If the response is valid and the authentication process has been completed, then a transition to the state Client Authenticated is made, the client is informed of its success by invoking authenticationSucceeded. In case the response is not valid, the Authentication object is destroyed. This implies that the client has to re-initiate the authentication by calling once more the initiateAuthenticationWithVersion method on the IpInitial interface. At any time the client may abort the authentication process by calling abortAuthentication on the Framework's IpAPILevelAuthentication interface. The client may also call selectAuthenticationMechanism to choose another hash algorithm.

6.4.1.2.10 Client Authenticated State

In this state the client is considered authenticated and is now allowed to request access to the IpAccess interface. If the framework decides to re-authenticate the client, then the challenge is sent to the client and a transition back to the AuthenticatingClient state occurs. The client may also call selectAuthenticationMechanism to choose another hash algorithm.

6.4.1.3 State Transition Diagrams for IpAccess

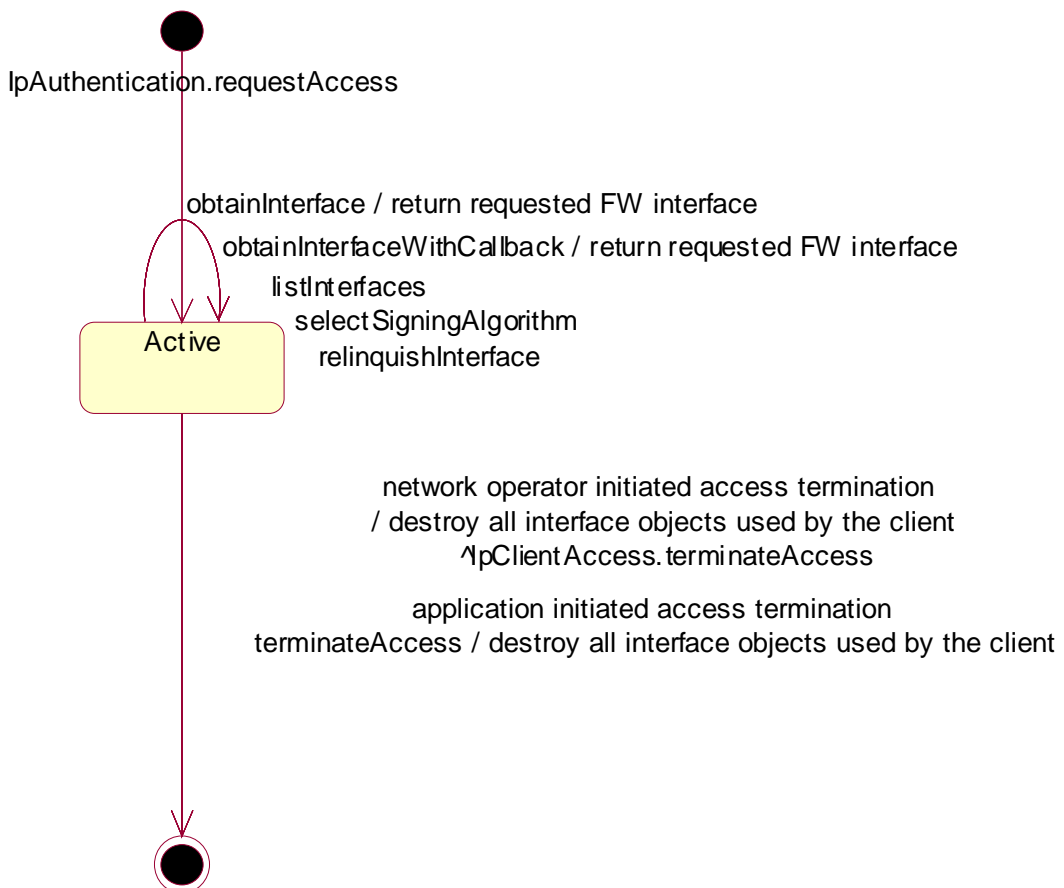


Figure : State Transition Diagram for IpAccess

6.4.1.3.1 Active State

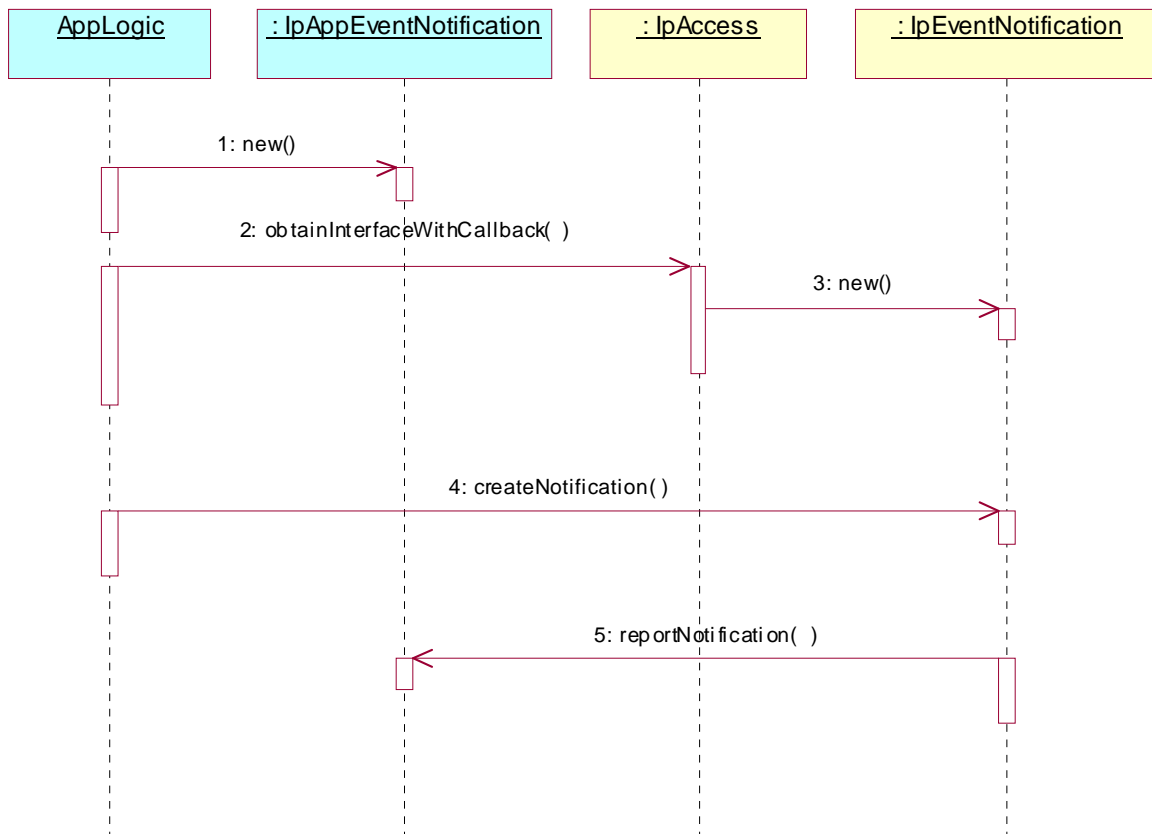
When the client requests access to the Framework on the IpAuthentication (IpAPILevelAuthentication) interface, an object implementing the IpAccess interface is created. The client can now request other Framework interfaces, including Service Discovery, Integrity Management, Service Subscription etc., and if at any point these framework interfaces are no longer required, to relinquish these. In addition the client can select the signing algorithm that shall be used during the access session in cases where a digital signature is required. When the client is no longer interested in using the interfaces it calls the terminateAccess method. This results in the destruction of all interface objects used by the client. In case the network operator decides that the client has no longer access to the interfaces the same will happen.

7 Framework-to-Application API

7.1 Sequence Diagrams

7.1.1 Event Notification Sequence Diagrams

7.1.1.1 Enable Event Notification



1: This message is used to create an object implementing the IpAppEventNotification interface.

2: This message is used to receive a reference to the object implementing the IpEventNotification interface and set the callback interface for the framework.

3: If there is currently no object implementing the IpEventNotification interface, then one is created using this message.

4: createNotification(eventCriteria : in TpFwEventCriteria) : TpAssignmentID.

This message is used to enable the notification mechanism so that subsequent framework events can be sent to the application. The framework event the application requests to be informed of is the availability of new SCFs.

Newly installed SCFs become available after the invocation of registerService and announceServiceAvailability on the Framework. The application uses the input parameter eventCriteria to specify the SCFs of whose availability it wants to be notified: those specified in ServiceTypeNameList.

The result of this invocation has many similarities with the result of invoking listServiceTypes: in both cases the application is informed of the availability of a list of SCFs. The differences are:

- in the case of invoking listServiceTypes, the application has to take the initiative, but it is informed of ALL SCFs available
- in the case of using the event notification mechanism, the application needs not take the initiative to ask about the availability of SCFs, but it is only informed of the ones that are newly available.

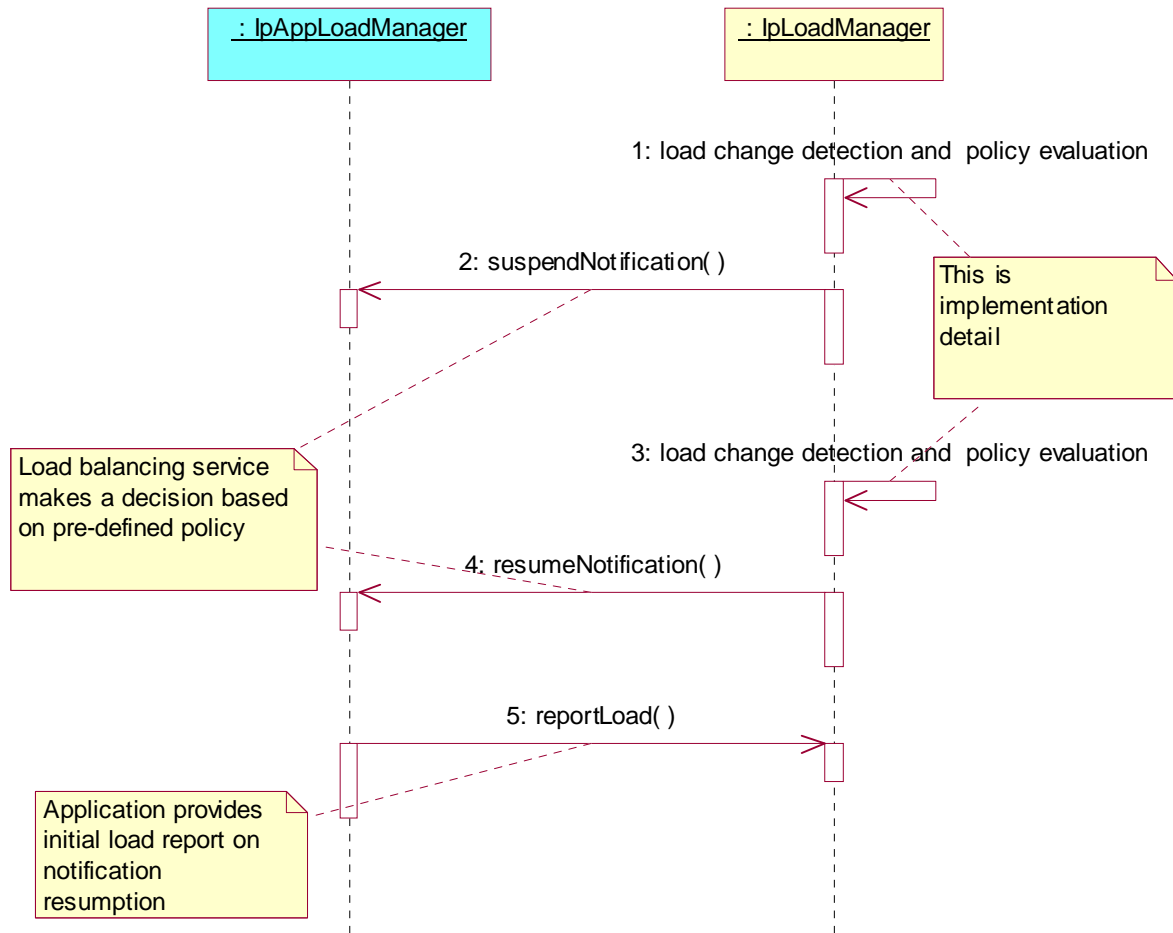
Alternatively, or additionally, the application can request to be informed of SCFs becoming unavailable.

5: The application is notified of the availability of new SCFs of the requested type(s).

7.1.2 Integrity Management Sequence Diagrams

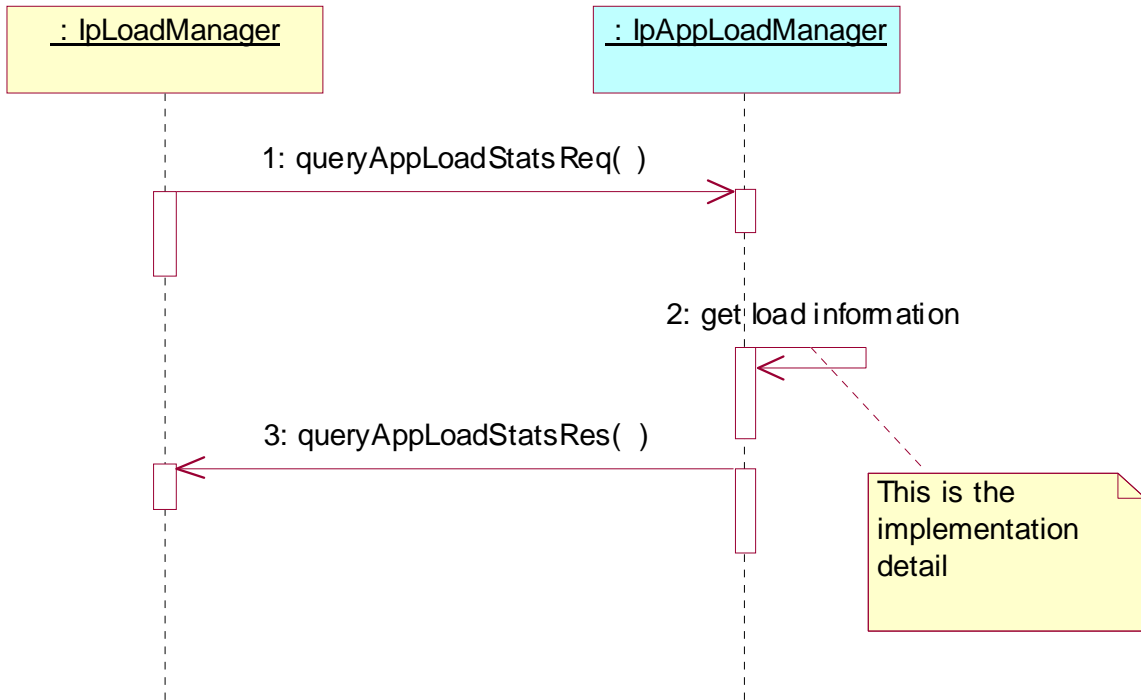
7.1.2.1 Load Management: Suspend/resume notification from application

This sequence diagram shows the scenario of suspending or resuming notifications from the application based on the evaluation of the load balancing policy as a result of the detection of a change in load level of the framework.



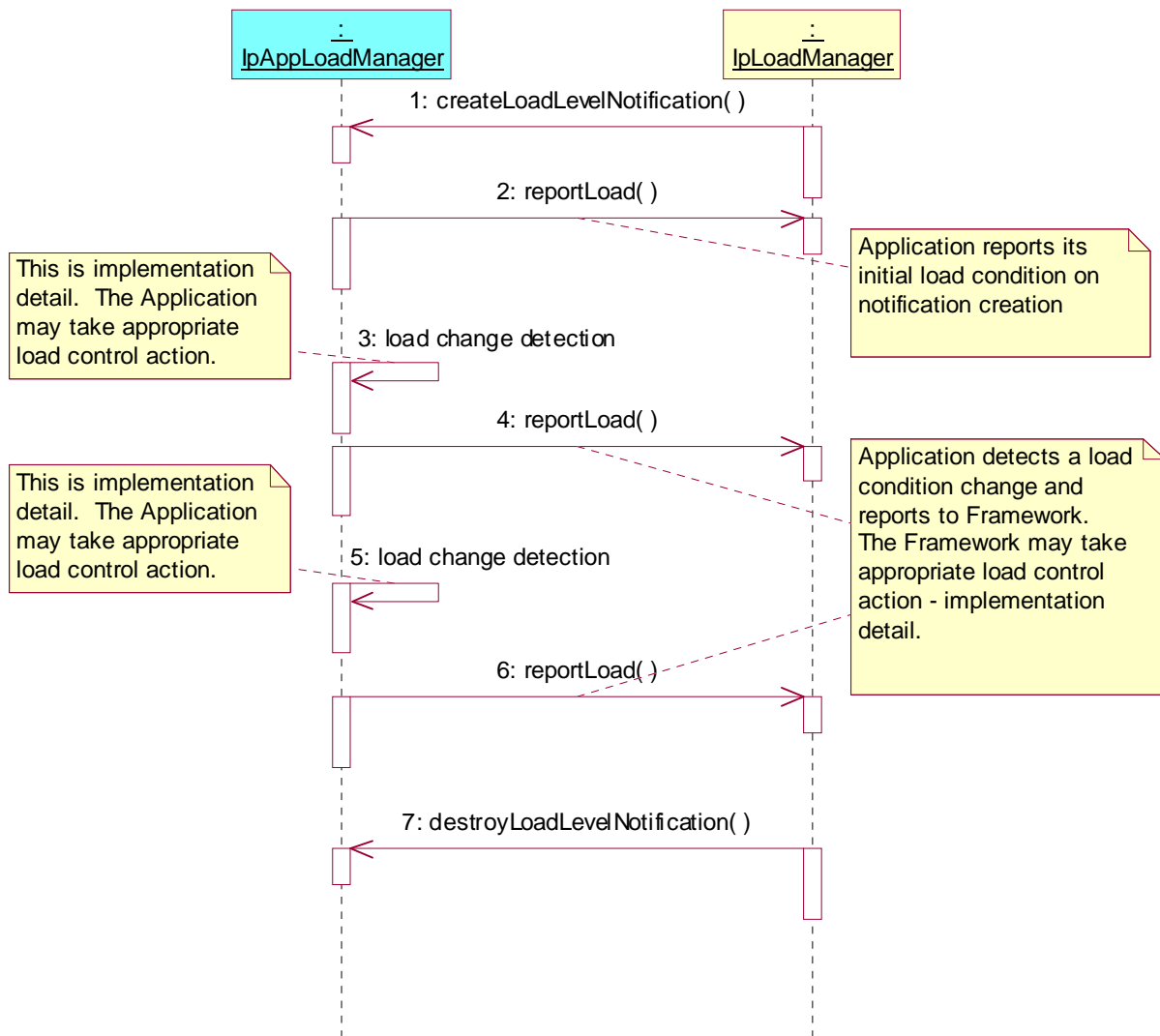
7.1.2.2 Load Management: Framework queries load statistics

This sequence diagram shows how the framework requests load statistics for an application.



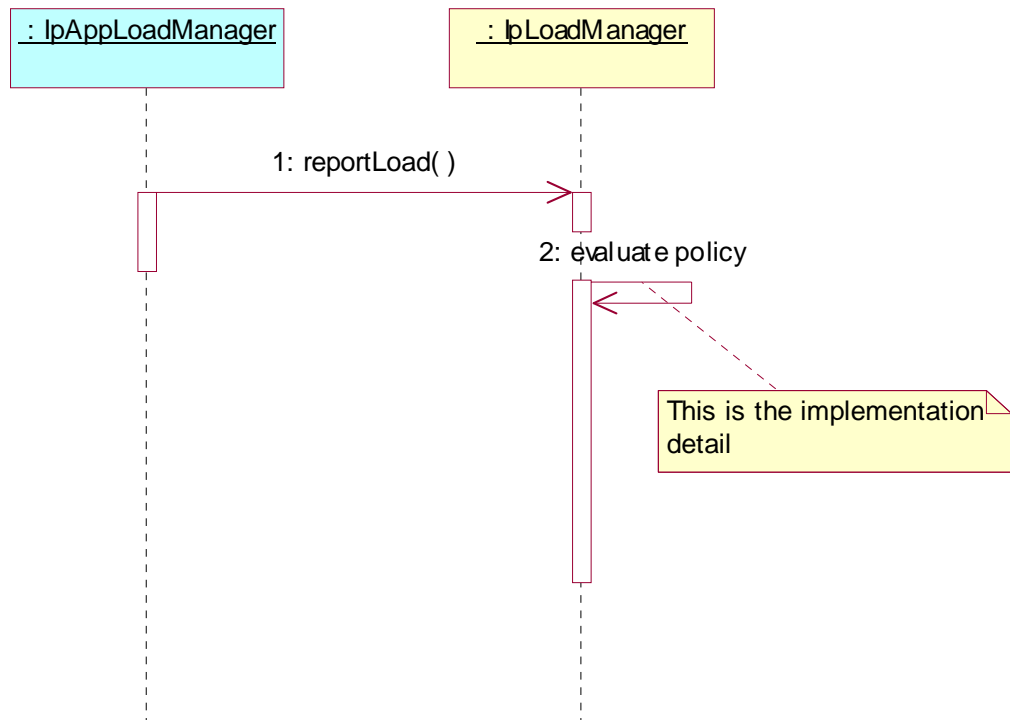
7.1.2.3 Load Management: Framework callback registration and Application load control

This sequence diagram shows how the framework registers itself and the application invokes load management function to inform the framework of application load.



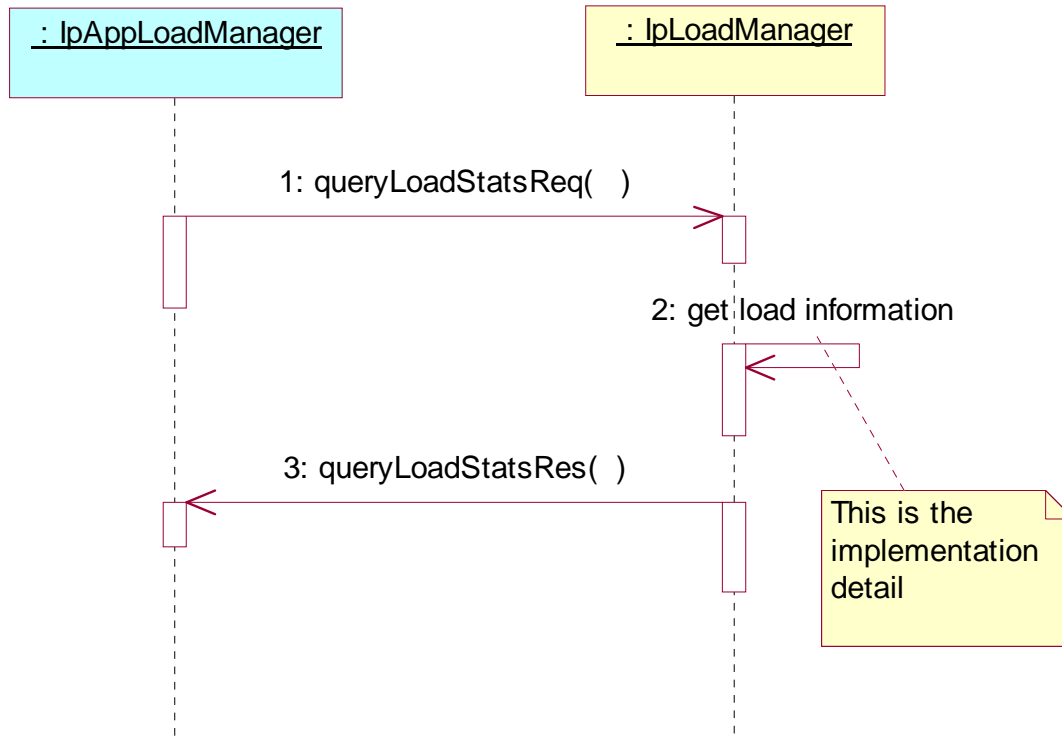
7.1.2.4 Load Management: Application reports current load condition

This sequence diagram shows how an application reports its load condition to the framework load manager.



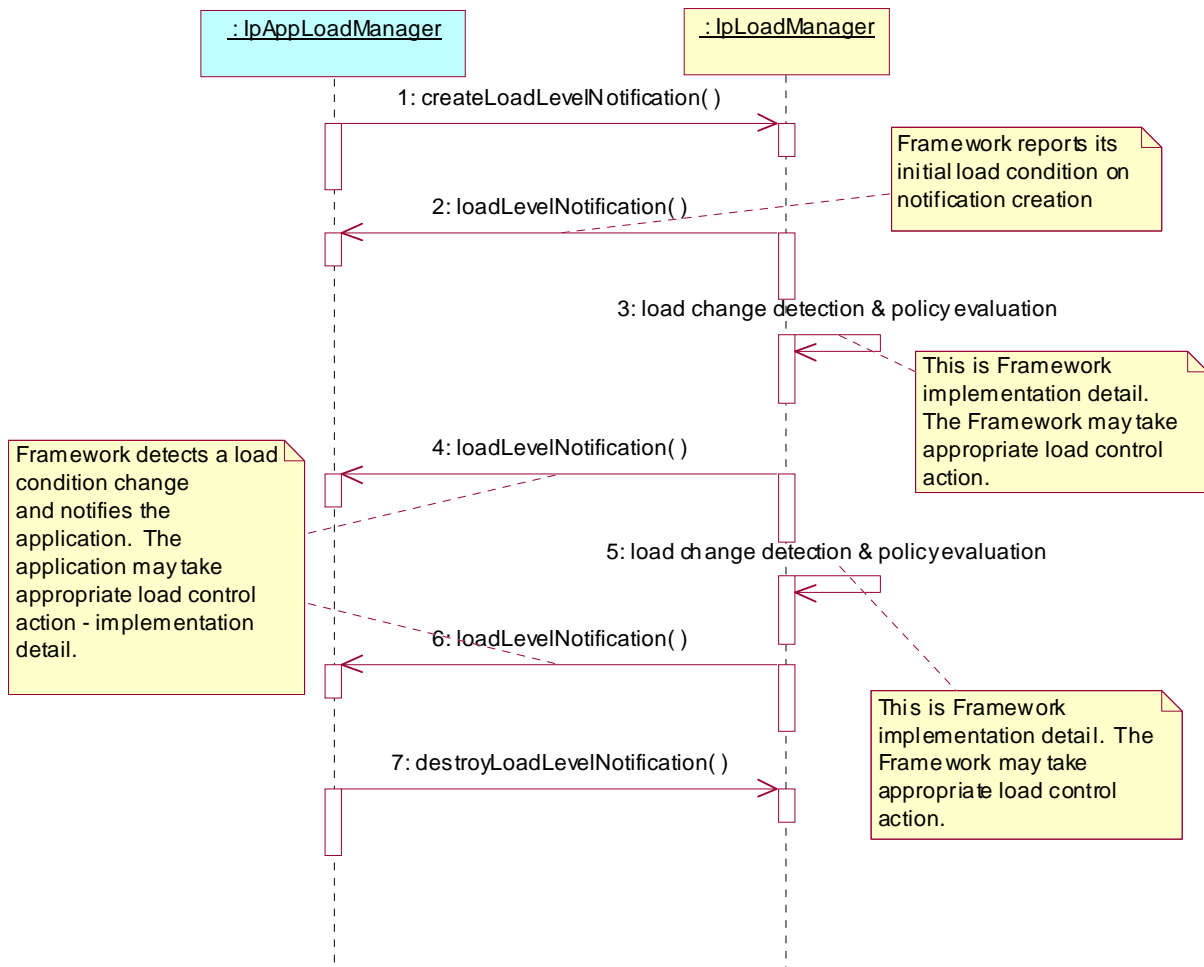
7.1.2.5 Load Management: Application queries load statistics

This sequence diagram shows how an application requests load statistics for the framework.



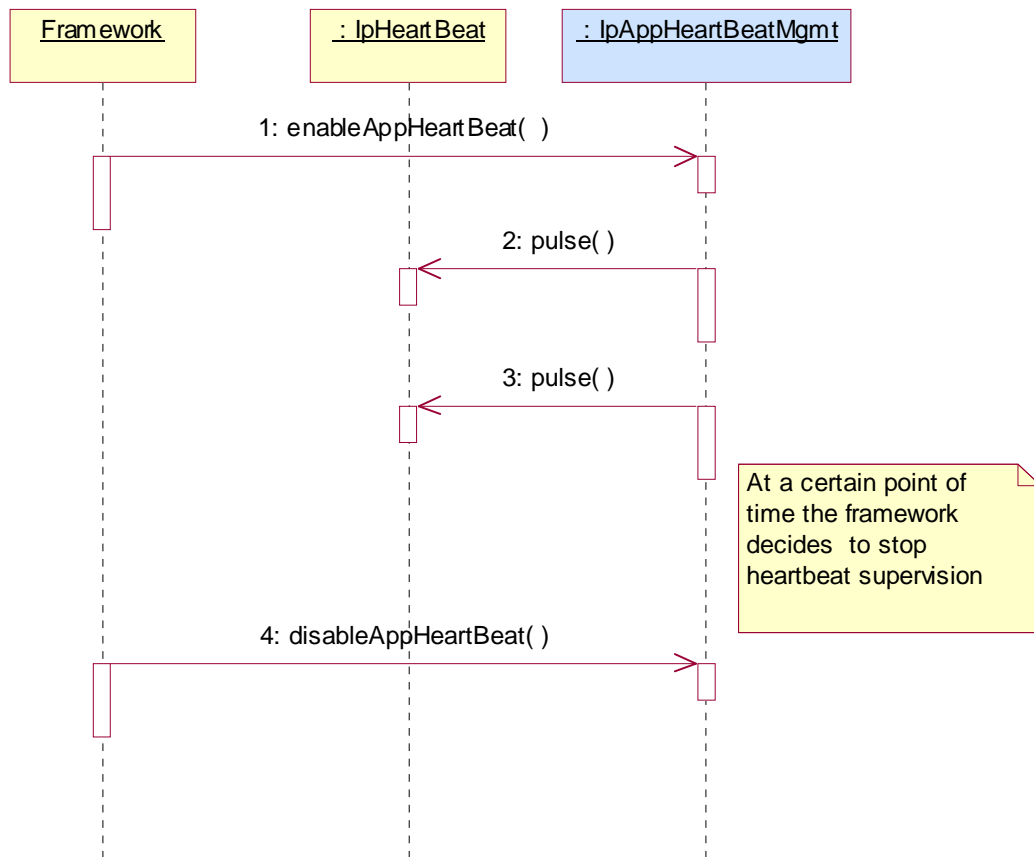
7.1.2.6 Load Management: Application callback registration and load control

This sequence diagram shows how an application registers itself and the framework invokes load management function based on policy.



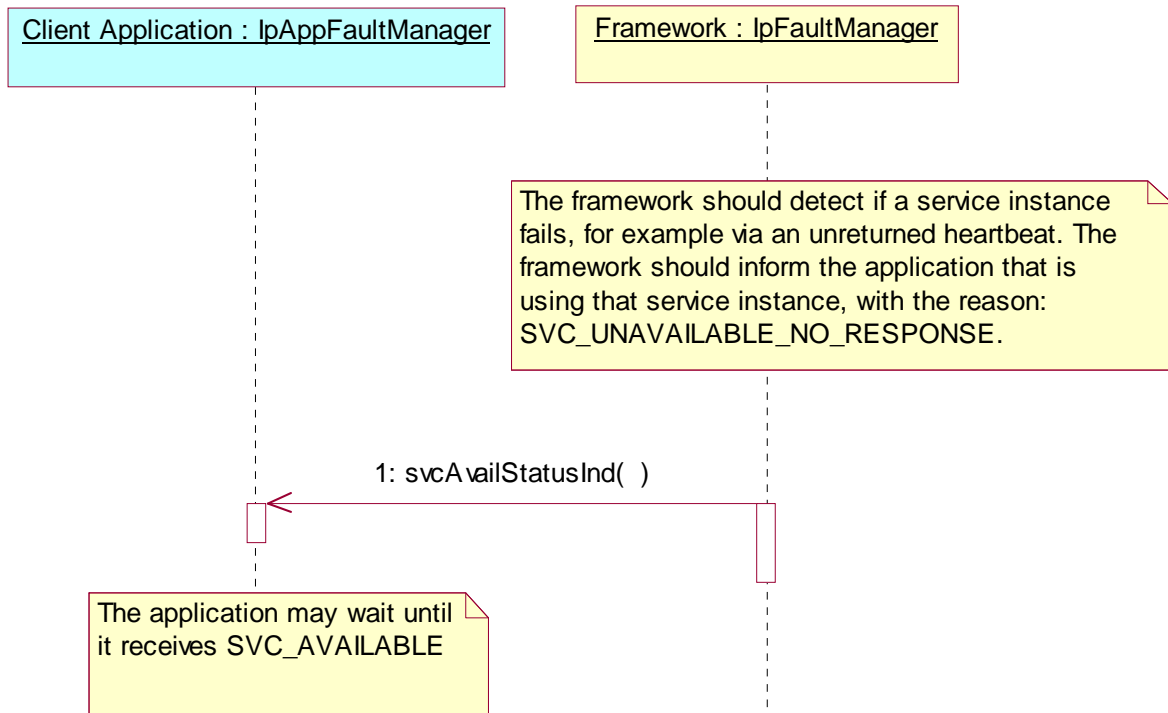
7.1.2.7 Heartbeat Management: Start/perform/end heartbeat supervision of the application

In this sequence diagram, the framework has decided that it wishes to monitor the application, and has therefore requested the application to commence sending its heartbeat. The application responds by sending its heartbeat at the specified interval. The framework then decides that it is satisfied with the application's health and disables the heartbeat mechanism. If the heartbeat was not received from the application within the specified interval, the framework can decide that the application has failed the heartbeat and can then perform some recovery action.



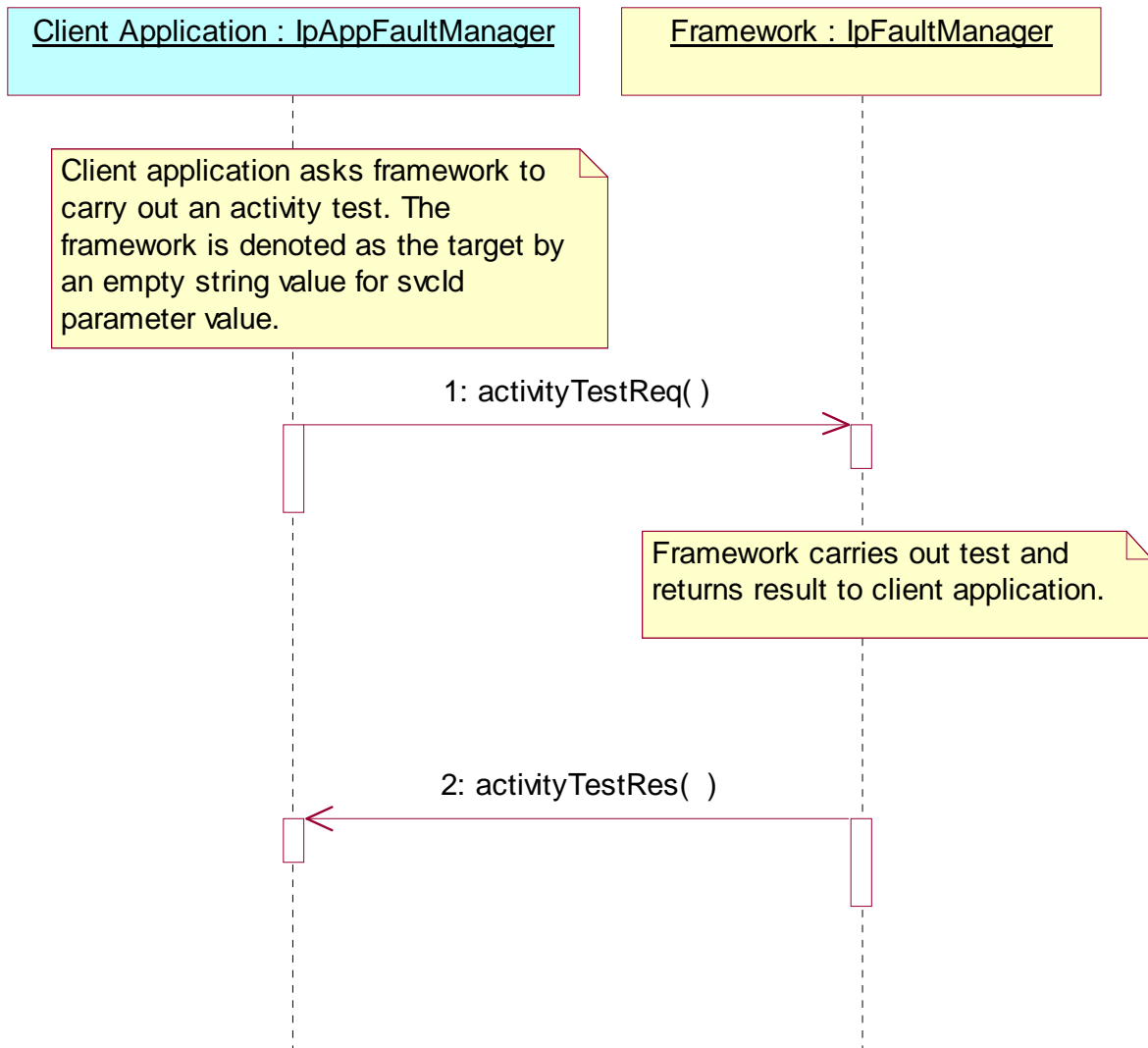
7.1.2.8 Fault Management: Framework detects a Service failure

The framework has detected that a service instance has failed (probably by the use of the heartbeat mechanism). The framework informs the client application.



1: The framework informs the client application that is using the service instance that the service is unavailable. The client application may wait to receive a new call to the svcAvailStatusInd with the reason SVC_AVAILABLE when the Service has become available again. The different Unavailability reasons used by the Framework (TpSvcAvailStatusReason) guides the client application developers to make the decision.

7.1.2.9 Fault Management: Application requests a Framework activity test



1: The client application asks the framework to do an activity test. The client identifies that it would like the activity test done for the framework, rather than a service, by supplying an empty string value for the svcId parameter.

2: The framework does the requested activity test and sends the result to the client application.

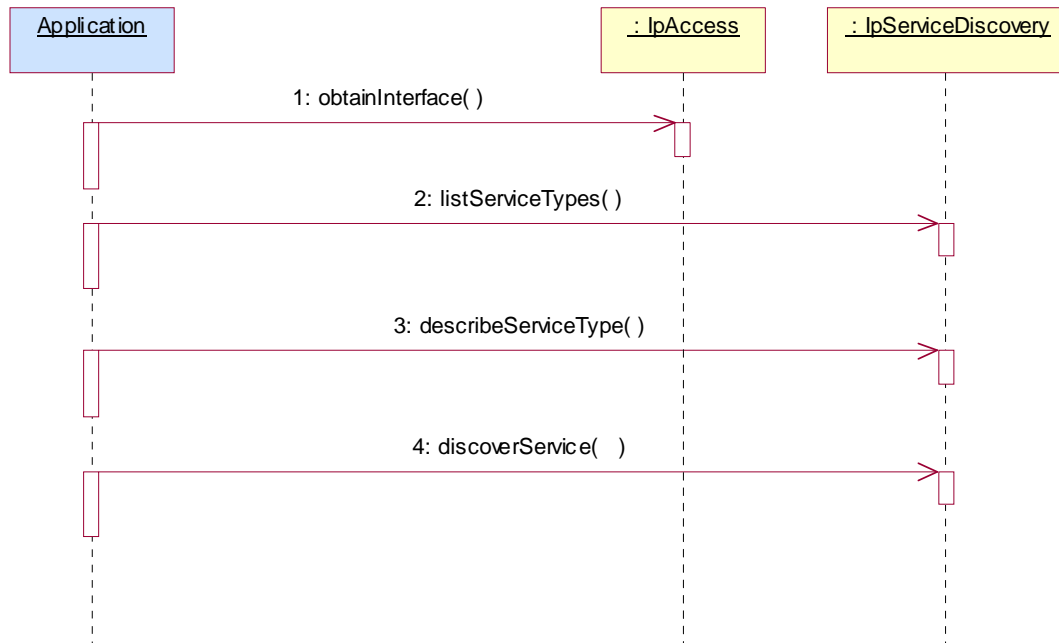
7.1.3 Service Discovery Sequence Diagrams

7.1.3.1 Service Discovery

The following figure shows how Applications discover a new Service Capability Feature in the network. Even applications that have already used the OSA API of a certain network know that the operator may upgrade it any time; this is why they use the Service Discovery interfaces.

Before the discovery process can start, the Application needs a reference to the Framework's Service Discovery interface; this is done via an invocation the method obtainInterface on the Framework's Access interface.

Discovery can be a three-step process. The first two steps have to be performed initially, but can subsequently be skipped (if the service type and its properties are already known, the application can invoke `discoverService()` without having to re-invoke the `list/discoverServiceType` methods).



2: Discovery: first step - list service types.

In this first step the application asks the Framework what service types that are available from this network. Service types are standardized or non-standardised SCF names, and thus this first step allows the Application to know what SCFs are supported by the network.

The following output is the result of this first discovery step:

- out listTypes.

This is a list of service type names, i.e., a list of strings, each of them the name of a SCF or a SCF specialization (e.g. "P_MPCC").

3: Discovery: second step - describe service type.

In this second step the application requests what are the properties that describe a certain service type that it is interested in, among those listed in the first step.

The following input is necessary:

- in name.

This is a service type name: a string that contains the name of the SCF whose description the Application is interested in (e.g. "P_MPCC").

And the output is:

- out serviceTypeDescription.

The description of the specified SCF type. The description provides information about:

- the property names associated with the SCF;
- the corresponding property value types;

- the corresponding property mode (mandatory or read only) associated with each SCF property;
- the names of the super types of this type; and
- whether the type is currently enabled or disabled.

4: Discovery: third step - discover service

In this third step the application requests for a service that matches its needs by tuning the service properties (i.e. assigning values for certain properties).

The Framework then checks whether there is a match, in which case it sends the Application the serviceID that is the identifier this network operator has assigned to the SCF version described in terms of those service properties. This is the moment where the serviceID identifier is shared with the application that is interested on the corresponding service.

This is done for either one service or more (the application specifies the maximum number of responses it wishes to accept).

Input parameters are:

- in serviceTypeName.

This is a string that contains the name of the SCF whose description the Application is interested in (e.g. "P_MPCC").

- in desiredPropertyList.

This is again a list like the one used for service registration, but where the value of the service properties have been fine tuned by the Application to (they will be logically interpreted as "minimum", "maximum", etc. by the Framework).

The following parameter is necessary as input:

- in max.

This parameter states the maximum number of SCFs that are to be returned in the "ServiceList" result.

And the output is:

- out serviceList.

This is a list of duplets: (serviceID, servicePropertyList). It provides a list of SCFs matching the requirements from the Application, and about each: the identifier that has been assigned to it in this network (serviceID), and once again the service property list.

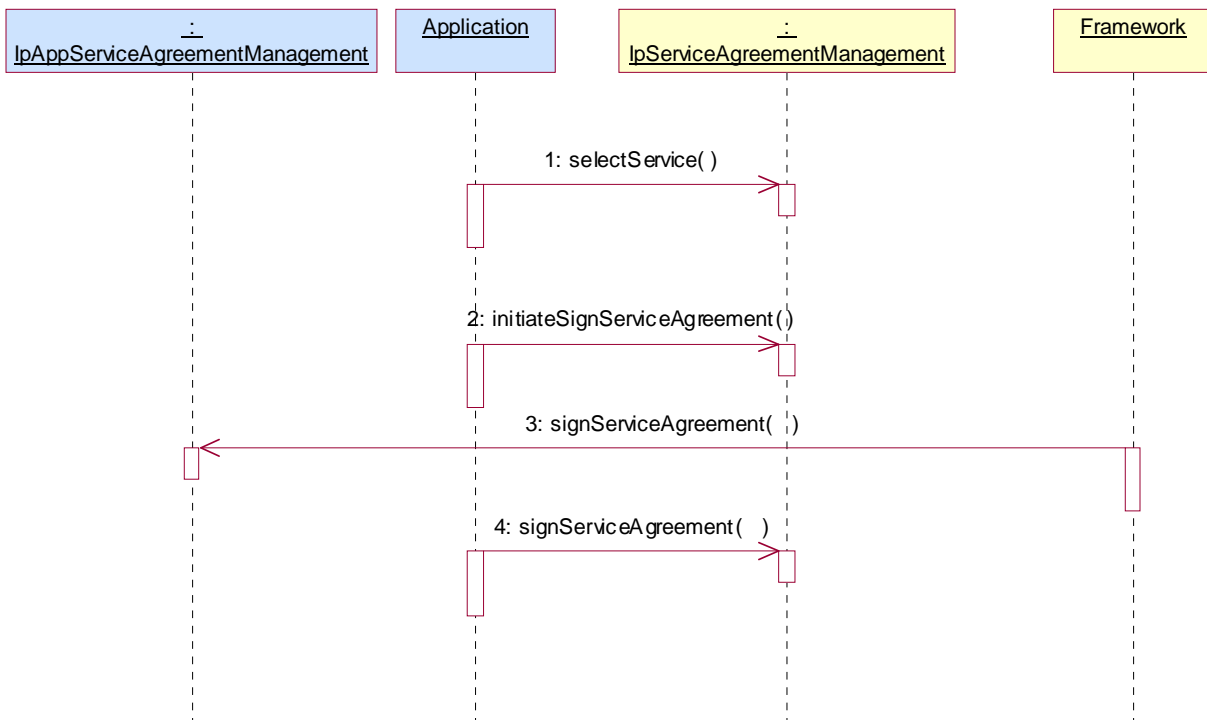
7.1.4 Service Agreement Management Sequence Diagrams

7.1.4.1 Service Selection

The following figure shows the process of selecting an SCF.

After discovery the Application gets a list of one or more SCF versions that match its required description. It now needs to decide which service it is going to use; it also needs to actually get a way to use it.

This is achieved by the following two steps:



1: Service Selection: first step - selectService

In this first step the Application identifies the SCF version it has finally decided to use. This is done by means of the serviceID, which is the agreed identifier for SCF versions. The Framework acknowledges this selection by returning to the Application an identifier for the service chosen: a service token, that is a private identifier for this service between this Application and this network, and is used for the process of signing the service agreement.

Input is:

- in serviceID.

This identifies the SCF required.

And output:

- out serviceToken.

This is a free format text token returned by the framework, which can be signed as part of a service agreement. It contains operator specific information relating to the service level agreement. An application (identifiable by a given TpClientAppID) may select the same service on more than one occasion in which case the same serviceToken, that identifies the relationship between the Application and the network, and the service agreement that applies, shall be returned.

2: Service Selection: second step - signServiceAgreement

In this second step an agreement is signed that allows the Application to use the chosen SCF version. And once these contractual details have been agreed, then the Application can be given the means to actually use it. The means are a reference to the manager interface of the SCF version (remember that a manager is an entry point to any SCF). By calling the createServiceManager operation on the lifecycle manager the Framework retrieves this interface and returns it to the Application. The service properties suitable for this application are also fed to the SCF (via the lifecycle manager interface) in order for the SCS to instantiate an SCF version that is suitable for this application.

The sequence of events indicated above, where the application initiates the signature process by calling initiateSignServiceAgreement, and where the framework calls signServiceAgreement on the application's IpAppServiceAgreementManagement interface before the application calls signServiceAgreement on the frameworks' IpServiceAgreementManagement, is the only sequence permitted.

Input:

- in serviceToken.

This is the identifier that the network and Application have agreed to privately use for a certain version of SCF.

- in agreementText.

This is the agreement text that is to be signed by the Framework using the private key of the Framework.

- in signingAlgorithm.

This is the algorithm used to compute the digital signature.

Output:

- out signatureAndServiceMgr.

This is a reference to a structure containing the digital signature of the Framework for the service agreement, and a reference to the manager interface of the SCF.

There must be only one service instance per client application. Therefore, in case an application (identifiable by a given TpClientAppID) attempts to select a service for which it has already signed a service agreement and this service agreement has not been terminated, the Framework may return a reference to the already existing service, or may raise an exception to the client indicating that this request is denied.

7.2 Class Diagrams

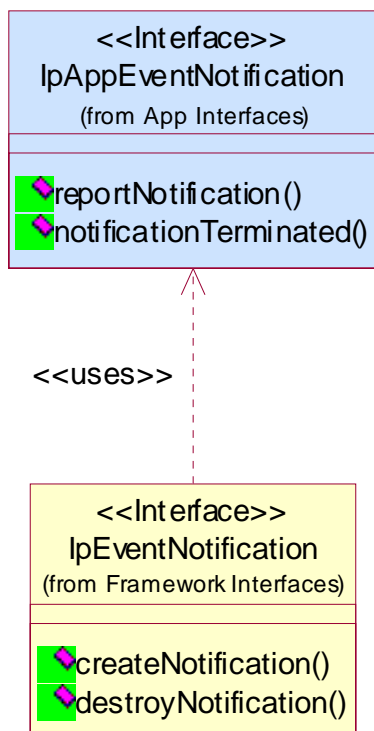


Figure: Event Notification Class Diagram

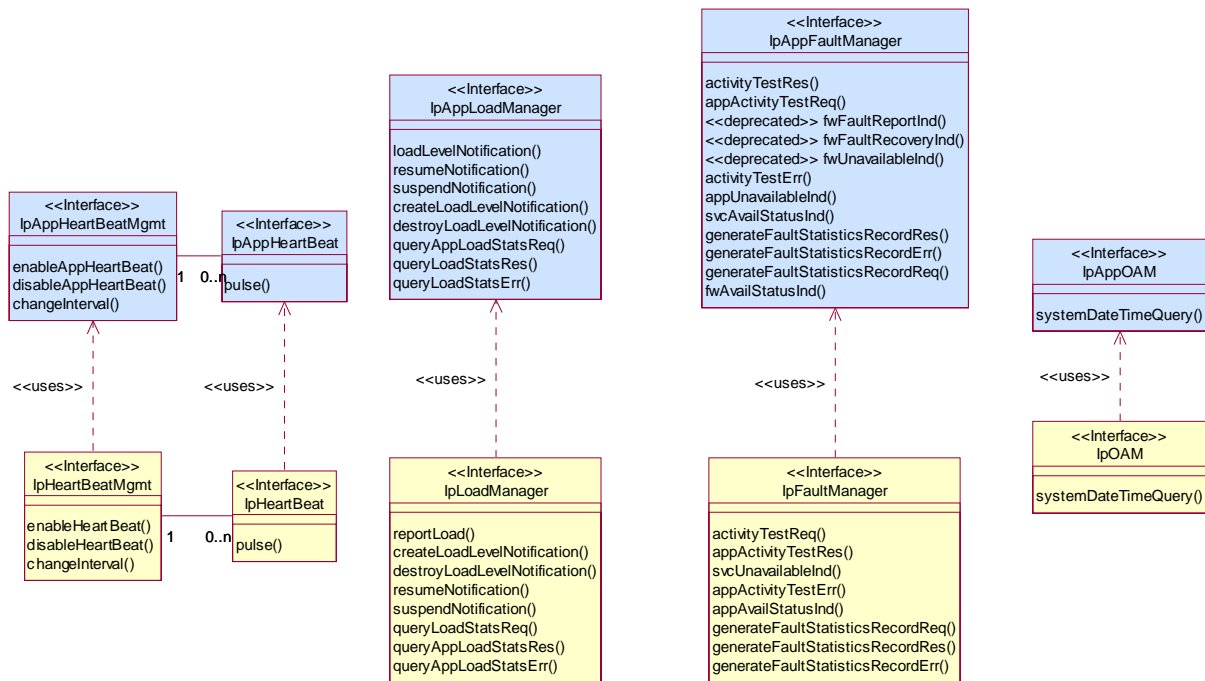


Figure: Integrity Management Package Overview

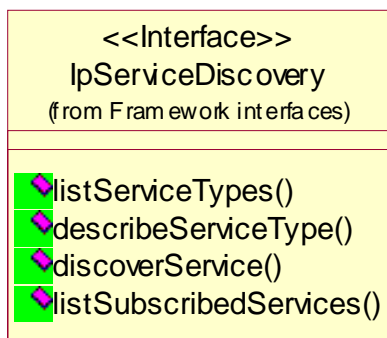


Figure: Service Discovery Package Overview

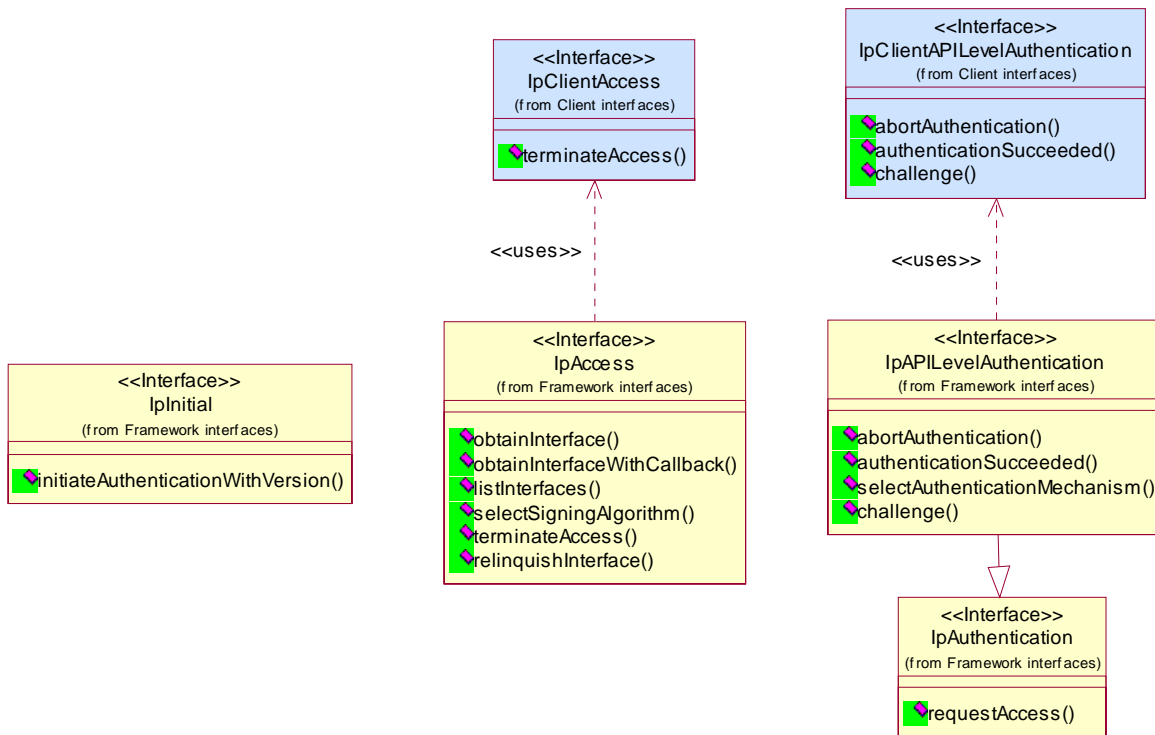


Figure: Trust and Security Management Package Overview

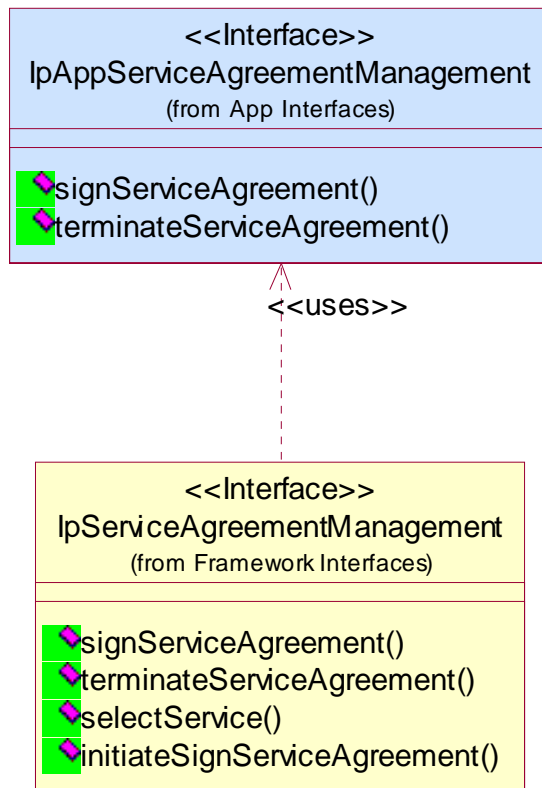


Figure: Service Agreement Management Package Overview

7.3 Interface Classes

7.3.1 Service Discovery Interface Classes

7.3.1.1 Interface Class IpServiceDiscovery

Inherits from: IpInterface.

The service discovery interface, shown below, consists of four methods. Before a service can be discovered, the enterprise operator (or the client applications) must know what "types" of services are supported by the Framework and what service "properties" are applicable to each service type. The listServiceTypes() method returns a list of all "service types" that are currently supported by the framework and the "describeServiceType()" returns a description of each service type. The description of service type includes the "service-specific properties" that are applicable to each service type. Then the enterprise operator (or the client applications) can discover a specific set of registered services that both belong to a given type and possess the desired "property values", by using the "discoverService() method. Once the enterprise operator finds out the desired set of services supported by the framework, it subscribes to (a sub-set of) these services using the Subscription Interfaces. The enterprise operator (or the client applications in its domain) can find out the set of services available to it (i.e., the service that it can use) by invoking "listSubscribedServices()". The service discovery APIs are invoked by the enterprise operators or client applications. They are described below.

This interface shall be implemented by a Framework with as a minimum requirement the listServiceTypes(), describeServiceType() and discoverService() methods.

<<Interface>> IpServiceDiscovery
<pre> listServiceTypes () : TpServiceTypeNameList describeServiceType (name : in TpServiceTypeName) : TpServiceTypeDescription discoverService (serviceName : in TpServiceTypeName, desiredPropertyList : in TpServicePropertyList, max : in TpInt32) : TpServiceList listSubscribedServices () : TpServiceList </pre>

7.3.1.1.1 Method listServiceTypes()

This operation returns the names of all service super and sub types that are in the repository. The details of the service types can then be obtained using the describeServiceType() method. If a sub type of a service is registered, this method returns, besides the sub type, also the super type.

Returns <listTypes> : The names of the requested service types.

Parameters

No Parameters were identified for this method.

Returns

TpServiceTypeNameList

Raises

TpCommonExceptions, P_ACCESS_DENIED

7.3.1.1.2 Method describeServiceType()

This operation lets the caller obtain the details for a particular service type.

Returns <serviceTypeDescription> : The description of the specified service type. The description provides information about:

- the service properties associated with this service type: i.e. a list of service property {name, mode and type} tuples;
- the names of the super types of this service type; and
- whether the service type is currently available or unavailable.

Parameters

name : in TpServiceTypeName

The name of the service type to be described.

- If the "name" is malformed, then the P_ILLEGAL_SERVICE_TYPE exception is raised.
- If the "name" does not exist in the repository, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

Returns

TpServiceTypeDescription

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE

7.3.1.1.3 Method discoverService()

The discoverService operation is the means by which a client application is able to obtain the service IDs of the services that meet its requirements. The client application passes in a list of desired service properties to describe the service it is looking for, in the form of attribute/value pairs for the service properties. The client application also specifies the maximum number of matched responses it is willing to accept. The framework must not return more matches than the specified maximum, but it is up to the discretion of the Framework implementation to choose to return less than the specified maximum. The discoverService() operation returns a serviceID/Property pair list for those services that match the desired service property list that the client application provided. The service properties returned form a complete view of what the client application can do with the service, as per the service level agreement. If the framework supports service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties.

Returns <serviceList> : This parameter gives a list of matching services. Each service is characterised by its service ID and a list of service properties {name and value list} associated with the service.

Parameters

serviceTypeName : in TpServiceTypeName

The "serviceTypeName" parameter conveys the required service type. It is key to the central purpose of "service trading". It is the basis for type safe interactions between the service exporters (via registerService) and service importers (via discoverService). By stating a service type, the importer implies the service type and a domain of discourse for talking about properties of service.

- If the string representation of the "type" does not obey the rules for service type identifiers, then the P_ILLEGAL_SERVICE_TYPE exception is raised.
- If the "type" is correct syntactically but is not recognised as a service type within the Framework, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

The framework may return a service of a subtype of the "type" requested. The requestor may also request for a service of a specific subtype. The framework will not return the corresponding supertype(s) in this case.

desiredPropertyList : in TpServicePropertyList

The "desiredPropertyList" parameter is a list of service property {name, mode and value list} tuples that the discovered set of services should satisfy. These properties deal with the non-functional and non-computational aspects of the desired service. The property values in the desired property list must be logically interpreted as "minimum", "maximum", etc. by the framework (due to the absence of a Boolean constraint expression for the specification of the service criterion). It is suggested that, at the time of service registration, each property value be specified as an appropriate range of values, so that desired property values can specify an "enclosing" range of values to help in the selection of desired services.

The desiredPropertyList only contains service properties that are relevant for the application. If an application is not interested in the value of a certain service property, this service property shall not be included in the desiredPropertyList.

P_INVALID_PROPERTY is raised when an application includes an unknown service property name or invalid service property value.

max : in **TpInt32**

The "max" parameter states the maximum number of services that are to be returned in the "serviceList" result.

Returns

TpServiceList

Raises

TpCommonExceptions, **P_ACCESS_DENIED**, **P_ILLEGAL_SERVICE_TYPE**,
P_UNKNOWN_SERVICE_TYPE, **P_INVALID_PROPERTY**

7.3.1.1.4 Method listSubscribedServices()

Returns a list of services so far subscribed by the enterprise operator. The enterprise operator (or the client applications in the enterprise domain) can obtain a list of subscribed services that they are allowed to access.

Returns <serviceList> : The "serviceList" parameter returns a list of subscribed services. Each service is characterised by its service ID and a list of service properties {name and value list} associated with the service.

Parameters

No Parameters were identified for this method.

Returns

TpServiceList

Raises

TpCommonExceptions, **P_ACCESS_DENIED**

7.3.2 Service Agreement Management Interface Classes

7.3.2.1 Interface Class IpAppServiceAgreementManagement

Inherits from: IpInterface.

This interface and the signServiceAgreement() and terminateServiceAgreement() methods shall be implemented by an application.

<<Interface>> IpAppServiceAgreementManagement
signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm) : TpOctetSet terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpOctetSet) : void

7.3.2.1.1 Method signServiceAgreement()

Upon receipt of the initiateSignServiceAgreement() method from the client application, this method is used by the framework to request that the client application sign an agreement on the service. The framework provides the service agreement text for the client application to sign. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. If the client application agrees, it signs the service agreement, returning its digital signature to the framework.

Returns <digitalSignature> : This contains a CMS (Cryptographic Message Syntax) object (as defined in RFC 2630) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is the agreement text given by the framework. The "external signature" construct shall not be used (i.e. the eContent field in the EncapsulatedContentInfo field shall be present and contain the agreement text). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. If the signature is incorrect the serviceToken will be expired immediately.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance to which this service agreement corresponds. (If the client application selects many services, it can determine which selected service corresponds to the service agreement by matching the service token). If the serviceToken is invalid, or not known by the client application, then the P_INVALID_SERVICE_TOKEN exception is thrown.

agreementText : in TpString

This is the agreement text that is to be signed by the client application using the private key of the client application. If the agreementText is invalid, then the P_INVALID_AGREEMENT_TEXT exception is thrown.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. It shall be identical to the one chosen by the framework in response to IpAccess.selectSigningAlgorithm(). If the signingAlgorithm is not the chosen one, is invalid, or unknown to the client application, the P_INVALID_SIGNING_ALGORITHM exception is thrown. The list of possible algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

Returns

TpOctetSet

Raises

TpCommonExceptions, P_INVALID_AGREEMENT_TEXT, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM

7.3.2.1.2 Method terminateServiceAgreement()

This method is used by the framework to terminate an agreement for the service.

Parameters

serviceToken : in TpServiceToken

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or unknown to the client application, the P_INVALID_SERVICE_TOKEN exception will be thrown.

terminationText : in TpString

This is the termination text that describes the reason for the termination of the service agreement.

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in RFC 2630) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630 using the same signing algorithm as was used to initially sign the service agreement. The content is the termination text. The "external signature" construct shall not be used (i.e. the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the service agreement is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

7.3.2.2 Interface Class IpServiceAgreementManagement

Inherits from: IpInterface.

This interface and the signServiceAgreement(), terminateServiceAgreement(), selectService() and initiateSignServiceAgreement() methods shall be implemented by a Framework.

<<Interface>> IpServiceAgreementManagement
<p>signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm) : TpSignatureAndServiceMgr</p> <p>terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpOctetSet) : void</p> <p>selectService (serviceID : in TpServiceID) : TpServiceToken</p> <p>initiateSignServiceAgreement (serviceToken : in TpServiceToken) : void</p>

7.3.2.2.1 Method signServiceAgreement()

After the framework has called signServiceAgreement() on the application's IpAppServiceAgreementManagement interface, this method is used by the client application to request that the framework sign the service agreement, which allows the client application to use the service. A reference to the service manager interface of the service is returned to the client application. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. If the client application is not allowed to access the service, then an error code (P_SERVICE_ACCESS_DENIED) is returned. If the client application invokes this method before the processing (i.e. digital signature verification) of the response of signServiceAgreement() on the application's IpAppServiceAgreementManagement interface has completed, a TpCommonExceptions with ExceptionType P_INVALID_STATE may be raised to indicate that this method is currently unable to complete the method due to a race condition. In this case, the TpCommonExceptions with ExceptionType P_INVALID_STATE suggests the application to retry the method invocation after a reasonable amount of time has passed.

There must be only one service instance per client application. Therefore, in case the client attempts to select a service for which it has already signed a service agreement and this service agreement has not been terminated, a reference to the already existing service manager will be returned.

Returns <signatureAndServiceMgr> : This contains the digital signature of the framework for the service agreement, and a reference to the service manager interface of the service.

```
structure TpSignatureAndServiceMgr {
    digitalSignature: TpOctetSet;
    serviceMgrInterface: IpServiceRef;
};
```

The digitalSignature contains a CMS (Cryptographic Message Syntax) object (as defined in RFC 2630) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is the agreement text given by the client application. The "external signature" construct shall not be used (i.e. the eContent field in the EncapsulatedContentInfo field shall be present and contain the agreement text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention.

The serviceMgrInterface is a reference to the service manager interface for the selected service.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance requested by the client application. If the serviceToken is invalid, or has expired, an error code (P_INVALID_SERVICE_TOKEN) is returned.

agreementText : in TpString

This is the agreement text that is to be signed by the framework using the private key of the framework. If the agreementText is invalid, then an error code (P_INVALID_AGREEMENT_TEXT) is returned.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. It shall be identical to the one used by the framework when invoking signServiceAgreement() on the client. If the signingAlgorithm is not the same one, is invalid, or unknown to the framework, an error code (P_INVALID_SIGNING_ALGORITHM) is returned. The list of possible algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

Returns

TpSignatureAndServiceMgr

Raises

**TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_AGREEMENT_TEXT,
P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM,
P_SERVICE_ACCESS_DENIED**

7.3.2.2.2 Method terminateServiceAgreement()

This method is used by the client application to terminate an agreement for the service.

Parameters

serviceToken : in TpServiceToken

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or has expired, an error code (P_INVALID_SERVICE_TOKEN) is returned.

terminationText : in TpString

This is the termination text that describes the reason for the termination of the service agreement.

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in RFC 2630) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630 using the same signing algorithm as was used to initially sign the service agreement. The content is the termination text. The "external signature" construct shall not be used (i.e. the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to check that the terminationText has been signed by the client application. If a match is made, the service agreement is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Raises

**TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_TOKEN,
P_INVALID_SIGNATURE**

7.3.2.2.3 Method selectService()

This method is used by the client application to identify the service that the client application wishes to use. If the client application is not allowed to access the service, then the P_SERVICE_ACCESS_DENIED exception is thrown.

Returns <serviceToken> : This is a free format text token returned by the framework, which can be signed as part of a service agreement. This will contain operator specific information relating to the service level agreement. The serviceToken has a limited lifetime. If the lifetime of the serviceToken expires, a method accepting the serviceToken will return an error code (P_INVALID_SERVICE_TOKEN). Service Tokens will automatically expire if the client application or framework invokes the terminateAccess method on the other's corresponding access interface.

*Parameters***serviceID : in TpServiceID**

This identifies the service required. If the serviceID is not recognised by the framework, an error code (P_INVALID_SERVICE_ID) is returned.

Returns

TpServiceToken

Raises

**TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID,
P_SERVICE_ACCESS_DENIED**

7.3.2.2.4 Method initiateSignServiceAgreement()

This method is used by the client application to initiate the sign service agreement process. This method shall be invoked following the application's call to selectService(), and before the signing of the service agreement can take place. If the client application is not allowed to initiate the sign service agreement process, the exception (P_SERVICE_ACCESS_DENIED) is thrown.

*Parameters***serviceToken : in TpServiceToken**

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance requested by the client application. If the serviceToken is invalid, or has expired, the exception (P_INVALID_SERVICE_TOKEN) is thrown.

*Raises***TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_SERVICE_ACCESS_DENIED**

7.3.3 Integrity Management Interface Classes

7.3.3.1 Interface Class IpAppFaultManager

Inherits from: IpInterface.

This interface is used to inform the application of events that affect the integrity of the Framework, Service or Client Application. The Fault Management Framework will invoke methods on the Fault Management Application Interface that is specified when the client application obtains the Fault Management interface: i.e. by use of the obtainInterfaceWithCallback operation on the IpAccess interface

<<Interface>> IpAppFaultManager
activityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void appActivityTestReq (activityTestID : in TpActivityTestID) : void <<deprecated>> fwFaultReportInd (fault : in TpInterfaceFault) : void <<deprecated>> fwFaultRecoveryInd (fault : in TpInterfaceFault) : void <<deprecated>> fwUnavailableInd (reason : in TpFwUnavailReason) : void activityTestErr (activityTestID : in TpActivityTestID) : void appUnavailableInd (serviceID : in TpServiceID) : void svcAvailStatusInd (serviceID : in TpServiceID, reason : in TpSvcAvailStatusReason) : void generateFaultStatisticsRecordRes (faultStatsReqID : in TpFaultReqID, faultStatistics : in TpFaultStatsRecord, serviceIDs : in TpServiceIDList) : void generateFaultStatisticsRecordErr (faultStatsReqID : in TpFaultReqID, faultStatistics : in TpFaultStatsErrorList, serviceIDs : in TpServiceIDList) : void generateFaultStatisticsRecordReq (faultStatsReqID : in TpFaultReqID, timePeriod : in TpTimeInterval) : void fwAvailStatusInd (reason : in TpFwAvailStatusReason) : void

7.3.3.1.1 Method activityTestRes()

The framework uses this method to return the result of a client application-requested activity test.

*Parameters***activityTestID : in TpActivityTestID**

Used by the client application to correlate this response (when it arrives) with the original request.

activityTestResult : in TpActivityTestRes

The result of the activity test.

7.3.3.1.2 Method appActivityTestReq()

The framework invokes this method to test that the client application is operational. On receipt of this request, the application must carry out a test on itself, to check that it is operating correctly. The application reports the test result by invoking the appActivityTestRes method on the IpFaultManager interface.

*Parameters***activityTestID : in TpActivityTestID**

The identifier provided by the framework to correlate the response (when it arrives) with this request.

7.3.3.1.3 Method <<deprecated>> fwFaultReportInd()

This method is deprecated and will be removed in a later release. It is strongly recommended not to implement this method. The new method fwAvailStatusInd shall be used instead, using the new type of reason parameter to inform the Application the reason why the Framework is unavailable.

The framework invokes this method to notify the client application of a failure within the framework. The client application must not continue to use the framework until it has recovered (as indicated by a fwFaultRecoveryInd).

*Parameters***fault : in TpInterfaceFault**

Specifies the fault that has been detected by the framework.

7.3.3.1.4 Method <<deprecated>> fwFaultRecoveryInd()

This method is deprecated and will be removed in a later release. It is strongly recommended not to implement this method. The new method fwAvailStatusInd shall be used instead, using the new type of reason parameter to inform the Application when the Framework becomes available again.

The framework invokes this method to notify the client application that a previously reported fault has been rectified. The application may then resume using the framework.

*Parameters***fault : in TpInterfaceFault**

Specifies the fault from which the framework has recovered.

7.3.3.1.5 Method <<deprecated>> fwUnavailableInd()

This method is deprecated and will be removed in a later release. It is strongly recommended not to implement this method. The new method fwAvailStatusInd shall be used instead, using the new type of reason parameter to inform the Application the reason why the Framework is unavailable and also when the Framework becomes available again.

The framework invokes this method to inform the client application that it is no longer available.

Parameters

reason : in TpFwUnavailReason

Identifies the reason why the framework is no longer available.

7.3.3.1.6 Method activityTestErr()

The framework uses this method to indicate that an error occurred during an application-initiated activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the application to correlate this response (when it arrives) with the original request.

7.3.3.1.7 Method appUnavailableInd()

The framework invokes this method to indicate to the application that the service instance has detected that it is not responding.

Parameters

serviceID : in TpServiceID

Specifies the service for which the indication of unavailability was received.

7.3.3.1.8 Method svcAvailStatusInd()

The framework invokes this method to inform the client application about the Service instance availability status, i.e. that it can no longer use its instance of the indicated service according to the reason parameter but as well information when the Service Instance becomes available again. On receipt of this request, the client application either acts to reset its use of the specified service (using the normal mechanisms, such as the discovery and authentication interfaces, to stop use of this service instance and begin use of a different service instance). The client application can also wait for the problem to be solved and just stop the usage of the service instance until the svcAvailStatusInd() is called again with the reason SVC_AVAILABLE.

Parameters

serviceID : in TpServiceID

Identifies the affected service.

reason : in TpSvcAvailStatusReason

Identifies the reason why the service is no longer available or that it has become available again.

7.3.3.1.9 Method generateFaultStatisticsRecordRes()

This method is used by the framework to provide fault statistics to a client application in response to a generateFaultStatisticsRecordReq method invocation on the IpFaultManager interface.

Parameters

faultStatsReqID : in TpFaultReqID

Used by the client application to correlate this response (when it arrives) with the original request.

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

serviceIDs : in TpServiceIDList

Specifies the framework or services that are included in the general fault statistics record. If the serviceIDs parameter is an empty list, then the fault statistics are for the framework.

In the case where a list of services is present, this is an ordered list in which the location of the service in this list corresponds to the location of the related fault statistics in the TpFaultStatsRecord returned.

7.3.3.1.10 Method generateFaultStatisticsRecordErr()

This method is used by the framework to indicate an error fulfilling the request to provide fault statistics, in response to a generateFaultStatisticsRecordReq method invocation on the IpFaultManager interface.

*Parameters***faultStatsReqID : in TpFaultReqID**

Used by the client application to correlate this error (when it arrives) with the original request.

faultStatistics : in TpFaultStatsErrorList

The list of fault statistics errors returned.

serviceIDs : in TpServiceIDList

Specifies the framework or services that are included in the list of fault statistics errors returned. If the serviceIDs parameter is an empty list, then the fault statistics error relates to the framework.

In the case where a list of services is present, this is an ordered list in which the location of the service in this list corresponds to the location of the related fault statistics error in the TpFaultStatsErrorList returned.

7.3.3.1.11 Method generateFaultStatisticsRecordReq()

This method is used by the framework to solicit fault statistics from the client application, for example when the framework was asked for these statistics by a service instance by using the generateFaultStatisticsRecordReq operation on the IpFwFaultManager interface. On receipt of this request, the client application must produce a fault statistics record, for the application during the specified time interval, which is returned to the framework using the generateFaultStatisticsRecordRes operation on the IpFaultManager interface.

*Parameters***faultStatsReqID : in TpFaultReqID**

The identifier provided by the framework to correlate the response (when it arrives) with this request.

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. Supplying both a start time and stop time as empty strings leaves the time period to the discretion of the client application.

7.3.3.1.12 Method fwAvailStatusInd()

The framework invokes this method to inform the client application about the Framework availability status, i.e. that it can no longer use the Framework according to the reason parameter or that the Framework has become available again. The client application may wait for the problem to be solved and just stop the usage of the Framework until the fwAvailStatusInd() is called again with the reason FRAMEWORK_AVAILABLE.

*Parameters***reason : in TpFwAvailStatusReason**

Identifies the reason why the framework is no longer available or that it has become available again.

7.3.3.2 Interface Class IpFaultManager

Inherits from: IpInterface.

This interface is used by the application to inform the framework of events that affect the integrity of the framework and services, and to request information about the integrity of the system. The fault manager operations do not exchange callback interfaces as it is assumed that the client application supplies its Fault Management callback interface at the time it obtains the Framework's Fault Management interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.

If the IpFaultManager interface is implemented by a Framework, at least one of these methods shall be implemented. If the Framework is capable of invoking the IpAppFaultManager.appActivityTestReq() method, it shall implement appActivityTestRes() and appActivityTestErr() in this interface. If the Framework is capable of invoking IpAppFaultManager.generateFaultStatisticsRecordReq(), it shall implement generateFaultStatisticsRecordRes() and generateFaultStatisticsRecordErr() in this interface.

<<Interface>> IpFaultManager
activityTestReq (activityTestID : in TpActivityTestID, svcID : in TpServiceID) : void appActivityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void svcUnavailableInd (serviceID : in TpServiceID) : void appActivityTestErr (activityTestID : in TpActivityTestID) : void appAvailStatusInd (reason : in TpAppAvailStatusReason) : void generateFaultStatisticsRecordReq (faultStatsReqID : in TpFaultReqID, timePeriod : in TpTimeInterval, serviceIDs : in TpServiceIDList) : void generateFaultStatisticsRecordRes (faultStatsReqID : in TpFaultReqID, faultStatistics : in TpFaultStatsRecord) : void generateFaultStatisticsRecordErr (faultStatsReqID : in TpFaultReqID, faultStatisticsError : in TpFaultStatisticsError) : void

7.3.3.2.1 Method activityTestReq()

The application invokes this method to test that the framework or its instance of a service is operational. On receipt of this request, the framework must carry out a test on itself or on the client's instance of the specified service, to check that it is operating correctly. The framework reports the test result by invoking the activityTestRes method on the IpAppFaultManager interface. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

For security reasons the client application has access to the service ID rather than the service instance ID. However, as there is a one to one relationship between the client application and a service, i.e. there is only one service instance of the specified service per client application, it is the obligation of the framework to determine the service instance ID from the service ID.

*Parameters***activityTestID : in TpActivityTestID**

The identifier provided by the client application to correlate the response (when it arrives) with this request.

svcID : in TpServiceID

Identifies either the framework or a service for testing. The framework is designated by an empty string.

*Raises***TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE****7.3.3.2.2 Method appActivityTestRes()**

The client application uses this method to return the result of a framework-requested activity test.

*Parameters***activityTestID : in TpActivityTestID**

Used by the framework to correlate this response (when it arrives) with the original request.

activityTestResult : in TpActivityTestRes

The result of the activity test.

*Raises***TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID****7.3.3.2.3 Method svcUnavailableInd()**

This method is used by the client application to inform the framework that it can no longer use its instance of the indicated service (either due to a failure in the client application or in the service instance itself). On receipt of this request, the framework should take the appropriate corrective action.

*Parameters***serviceID : in TpServiceID**

Identifies the service that the application can no longer use.

*Raises***TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE****7.3.3.2.4 Method appActivityTestErr()**

The client application uses this method to indicate that an error occurred during a framework-requested activity test.

*Parameters***activityTestID : in TpActivityTestID**

Used by the framework to correlate this response (when it arrives) with the original request.

*Raises***TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID****7.3.3.2.5 Method appAvailStatusInd()**

This method is used by the application to inform the framework of its availability status. If the Application has detected a failure it uses one of the APP_UNAVAILABLE reason types to indicate the problem and that it is ceasing its use of all of its subscribed service instances. When the Application is working again it shall call this method again with the APP_AVAILABLE reason to inform the Framework that it is working properly again. The Framework shall also attempt to inform all of the service instances used by the specific application and/or its administrator about the problem.

*Parameters***reason : in TpAppAvailStatusReason**

Identifies the reason why the application is no longer available. APP_AVAILABLE is used to inform the Framework and the Service that the Application is available again.

*Raises***TpCommonExceptions****7.3.3.2.6 Method generateFaultStatisticsRecordReq()**

This method is used by the application to solicit fault statistics from the framework. On receipt of this request the framework must produce a fault statistics record, for either the framework or for the client's instances of the specified services during the specified time interval, which is returned to the client application using the generateFaultStatisticsRecordRes operation on the IpAppFaultManager interface. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

*Parameters***faultStatsReqID : in TpFaultReqID**

The identifier provided by the application to correlate the response (when it arrives) with this request.

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. Supplying both a start time and stop time as empty strings leaves the time period to the discretion of the framework.

serviceIDs : in TpServiceIDList

Specifies either the framework or services to be included in the general fault statistics record. If this parameter is not an empty list, the fault statistics records of the client's instances of the specified services are returned, otherwise the fault statistics record of the framework is returned.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

7.3.3.2.7 Method generateFaultStatisticsRecordRes()

This method is used by the client application to provide fault statistics to the framework in response to a generateFaultStatisticsRecordReq method invocation on the IpAppFaultManager interface.

Parameters

faultStatsReqID : in TpFaultReqID

Used by the framework to correlate this response (when it arrives) with the original request.

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

Raises

TpCommonExceptions

7.3.3.2.8 Method generateFaultStatisticsRecordErr()

This method is used by the client application to indicate an error fulfilling the request to provide fault statistics, in response to a generateFaultStatisticsRecordReq method invocation on the IpAppFaultManager interface.

Parameters

faultStatsReqID : in TpFaultReqID

Used by the framework to correlate this error (when it arrives) with the original request.

faultStatisticsError : in TpFaultStatisticsError

The fault statistics error.

Raises

TpCommonExceptions

7.3.3.3 Interface Class IpAppHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the client application by the framework.

<<Interface>> IpAppHeartBeatMgmt
enableAppHeartBeat (interval : in TpInt32, fwInterface : in IpHeartBeatRef) : void disableAppHeartBeat () : void changeInterval (interval : in TpInt32) : void

7.3.3.3.1 Method enableAppHeartBeat()

With this method, the framework instructs the client application to begin sending its heartbeat to the specified interface at the specified interval.

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

fwInterface : in IpHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

7.3.3.3.2 Method disableAppHeartBeat()

Instructs the client application to cease the sending of its heartbeat.

Parameters

No Parameters were identified for this method.

7.3.3.3.3 Method changeInterval()

Allows the administrative change of the heartbeat interval.

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

7.3.3.4 Interface Class IpAppHeartBeat

Inherits from: IpInterface.

The Heartbeat Application interface is used by the Framework to send the client application its heartbeat.

<<Interface>> IpAppHeartBeat
pulse () : void

7.3.3.4.1 Method pulse()

The framework uses this method to send its heartbeat to the client application. The application will be expecting a pulse at the end of every interval specified in the parameter to the IpHeartBeatMgmt.enableHeartbeat() method. If the pulse() is not received within the specified interval, then the framework can be deemed to have failed the heartbeat.

Parameters

No Parameters were identified for this method.

7.3.3.5 Interface Class IpHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the framework by a client application. If the IpHeartBeatMgmt interface is implemented by a Framework, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.

<<Interface>> IpHeartBeatMgmt
enableHeartBeat (interval : in TpInt32, appInterface : in IpAppHeartBeatRef) : void disableHeartBeat () : void changeInterval (interval : in TpInt32) : void

7.3.3.5.1 Method enableHeartBeat()

With this method, the client application instructs the framework to begin sending its heartbeat to the specified interface at the specified interval.

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

appInterface : in IpAppHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

Raises

TpCommonExceptions

7.3.3.5.2 Method disableHeartBeat()

Instructs the framework to cease the sending of its heartbeat.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

7.3.3.5.3 Method changeInterval()

Allows the administrative change of the heartbeat interval.

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

Raises

TpCommonExceptions

7.3.3.6 Interface Class IpHeartBeat

Inherits from: IpInterface.

The Heartbeat Framework interface is used by the client application to send its heartbeat. If a Framework is capable of invoking IpAppHeartBeatMgmt.enableHeartBeat(), it shall implement IpHeartBeat and the pulse() method.

<<Interface>> IpHeartBeat
pulse () : void

7.3.3.6.1 Method pulse()

The client application uses this method to send its heartbeat to the framework. The framework will be expecting a pulse at the end of every interval specified in the parameter to the IpAppHeartBeatMgmt.enableAppHeartbeat() method. If

the pulse() is not received within the specified interval, then the client application can be deemed to have failed the heartbeat.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

7.3.3.7 Interface Class IpAppLoadManager

Inherits from: IpInterface.

The client application developer supplies the load manager application interface to handle requests, reports and other responses from the framework load manager function. The application supplies the identity of this callback interface at the time it obtains the framework's load manager interface, by use of the obtainInterfaceWithCallback() method on the IpAccess interface.

<<Interface>> IpAppLoadManager
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void resumeNotification () : void suspendNotification () : void createLoadLevelNotification () : void destroyLoadLevelNotification () : void queryAppLoadStatsReq (loadStatsReqID : in TpLoadTestID, timeInterval : in TpTimeInterval) : void queryLoadStatsRes (loadStatsReqID : in TpLoadTestID, loadStatistics : in TpLoadStatisticList) : void queryLoadStatsErr (loadStatsReqID : in TpLoadTestID, loadStatisticsError : in TpLoadStatisticError) : void

7.3.3.7.1 Method loadLevelNotification()

Upon detecting load condition change, (e.g. load level changing from 0 to 1, 0 to 2, 1 to 0, for the SCFs or framework which have been registered for load level notifications) this method is invoked on the application. In addition this method shall be invoked on the application in order to provide a notification of current load status, when load notifications are first requested, or resumed after suspension.

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the framework-supplied load statistics, which include the load level change(s).

7.3.3.7.2 Method resumeNotification()

The framework uses this method to request the application to resume sending it notifications: e.g. after a period of suspension during which the framework handled a temporary overload condition. Upon receipt of this method the

client application shall inform the framework of the current load using the reportLoad method on the corresponding IpLoadManager.

Parameters

No Parameters were identified for this method.

7.3.3.7.3 Method suspendNotification()

The framework uses this method to request the application to suspend sending it any notifications: e.g. while the framework handles a temporary overload condition.

Parameters

No Parameters were identified for this method.

7.3.3.7.4 Method createLoadLevelNotification()

The framework uses this method to register to receive notifications of load level changes associated with the application. Upon receipt of this method the client application shall inform the framework of the current load using the reportLoad method on the corresponding IpLoadManager.

Parameters

No Parameters were identified for this method.

7.3.3.7.5 Method destroyLoadLevelNotification()

The framework uses this method to unregister for notifications of load level changes associated with the application.

Parameters

No Parameters were identified for this method.

7.3.3.7.6 Method queryAppLoadStatsReq()

The framework uses this method to request the application to provide load statistics records for the application.

Parameters

loadStatsReqID : in TpLoadTestID

The identifier provided by the framework to correlate the response (when it arrives) with this request.

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistic records should be reported.

7.3.3.7.7 Method queryLoadStatsRes()

The framework uses this method to send load statistic records back to the application that requested the information; i.e. in response to an invocation of the queryLoadStatsReq method on the IpLoadManager interface.

Parameters

loadStatsReqID : in TpLoadTestID

Used by the client application to correlate this response (when it arrives) with the original request.

loadStatistics : in TpLoadStatisticList

Specifies the framework-supplied load statistics.

7.3.3.7.8 Method queryLoadStatsErr()

The framework uses this method to return an error response to the application that requested the framework's load statistics information, when the framework is unsuccessful in obtaining any load statistic records; i.e. in response to an invocation of the queryLoadStatsReq method on the IpLoadManager interface.

Parameters

loadStatsReqID : in TpLoadTestID

Used by the client application to correlate this error (when it arrives) with the original request.

loadStatisticsError : in TpLoadStatisticError

Specifies the error code associated with the failed attempt to retrieve the framework's load statistics.

7.3.3.8 Interface Class IpLoadManager

Inherits from: IpInterface.

The framework API should allow the load to be distributed across multiple machines and across multiple component processes, according to a load management policy. The separation of the load management mechanism and load management policy ensures the flexibility of the load management services. The load management policy identifies what load management rules the framework should follow for the specific client application. It might specify what action the framework should take as the congestion level changes. For example, some real-time critical applications will want to make sure continuous service is maintained, below a given congestion level, at all costs, whereas other services will be satisfied with disconnecting and trying again later if the congestion level rises. Clearly, the load management policy is related to the QoS level to which the application is subscribed. The framework load management function is represented by the IpLoadManager interface. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. To handle responses and reports, the client application developer must implement the IpAppLoadManager interface to provide the callback mechanism. The application supplies the identity of this callback interface at the time it obtains the framework's load manager interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.

If the IpLoadManager interface is implemented by a Framework, at least one of the methods shall be implemented as a minimum requirement. If load level notifications are supported, the createLoadLevelNotification() and destroyLoadLevelNotification() methods shall be implemented. If suspendNotification() is implemented, then resumeNotification() shall be implemented also. If a Framework is capable of invoking the IpAppLoadManager.queryAppLoadStatsReq() method, then it shall implement queryAppLoadStatsRes() and queryAppLoadStatsErr() methods in this interface.

<<Interface>> IpLoadManager
<pre> reportLoad (loadLevel : in TpLoadLevel) : void createLoadLevelNotification (serviceIDs : in TpServiceIDList) : void destroyLoadLevelNotification (serviceIDs : in TpServiceIDList) : void resumeNotification (serviceIDs : in TpServiceIDList) : void suspendNotification (serviceIDs : in TpServiceIDList) : void queryLoadStatsReq (loadStatsReqID : in TpLoadTestID, serviceIDs : in TpServiceIDList, timeInterval : in TpTimeInterval) : void queryAppLoadStatsRes (loadStatsReqID : in TpLoadTestID, loadStatistics : in TpLoadStatisticList) : void queryAppLoadStatsErr (loadStatsReqID : in TpLoadTestID, loadStatisticsError : in TpLoadStatisticError) : void </pre>

7.3.3.8.1 Method reportLoad()

The client application uses this method to report its current load level (0, 1, or 2) to the framework: e.g. when the load level on the application has changed.

At level 0 load, the application is performing within its load specifications (i.e. it is not congested or overloaded). At level 1 load, the application is overloaded. At level 2 load, the application is severely overloaded. In addition this method shall be called by the application in order to report current load status, when load notifications are first requested, or resumed after suspension.

Parameters

loadLevel : in TpLoadLevel

Specifies the application's load level.

Raises

TpCommonExceptions

7.3.3.8.2 Method createLoadLevelNotification()

The client application uses this method to register to receive notifications of load level changes associated with either the framework or with its instances of the individual services used by the application. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID. Upon receipt of this method the framework shall inform the client application of the current framework or service instance load using the loadLevelNotification method on the corresponding IpAppLoadManager.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or SCFs to be registered for load control. To register for framework load control, the serviceIDs parameter must be an empty list.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

7.3.3.8.3 Method destroyLoadLevelNotification()

The client application uses this method to unregister for notifications of load level changes associated with either the framework or with its instances of the individual services used by the application. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which load level changes should no longer be reported. To unregister for framework load control, the serviceIDs parameter must be an empty list.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

7.3.3.8.4 Method resumeNotification()

The client application uses this method to request the framework to resume sending it load management notifications associated with either the framework or with its instances of the individual services used by the application; e.g. after a period of suspension during which the application handled a temporary overload condition. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID. Upon receipt of this method the framework shall inform the client application of the current framework or service instance load using the loadLevelNotification method on the corresponding IpAppLoadManager.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which the sending of notifications of load level changes by the framework should be resumed. To resume notifications for the framework, the serviceIDs parameter must be an empty list.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

7.3.3.8.5 Method suspendNotification()

The client application uses this method to request the framework to suspend sending it load management notifications associated with either the framework or with its instances of the individual services used by the application; e.g. while the application handles a temporary overload condition. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

*Parameters***serviceIDs : in TpServiceIDList**

Specifies the framework or the services for which the sending of notifications by the framework should be suspended. To suspend notifications for the framework, the serviceIDs parameter must be an empty list.

*Raises***TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED,
P_UNAUTHORISED_PARAMETER_VALUE****7.3.3.8.6 Method queryLoadStatsReq()**

The client application uses this method to request the framework to provide load statistic records for the framework or for its instances of the individual services. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

*Parameters***loadStatsReqID : in TpLoadTestID**

The identifier provided by the application to correlate the response (when it arrives) with this request.

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which load statistics records should be reported. If this parameter is not an empty list, the load statistics records of the client's instances of the specified services are returned, otherwise the load statistics record of the framework is returned.

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistics records should be reported.

*Raises***TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED,
P_UNAUTHORISED_PARAMETER_VALUE****7.3.3.8.7 Method queryAppLoadStatsRes()**

The client application uses this method to send load statistic records back to the framework that requested the information; i.e. in response to an invocation of the queryAppLoadStatsReq method on the IpAppLoadManager interface.

*Parameters***loadStatsReqID : in TpLoadTestID**

Used by the framework to correlate this response (when it arrives) with the original request.

loadStatistics : in TpLoadStatisticList

Specifies the application-supplied load statistics.

*Raises***TpCommonExceptions****7.3.3.8.8 Method queryAppLoadStatsErr()**

The client application uses this method to return an error response to the framework that requested the application's load statistics information, when the application is unsuccessful in obtaining any load statistic records; i.e. in response to an invocation of the queryAppLoadStatsReq method on the IpAppLoadManager interface.

Parameters

loadStatsReqID : in TpLoadTestID

Used by the framework to correlate this error (when it arrives) with the original request.

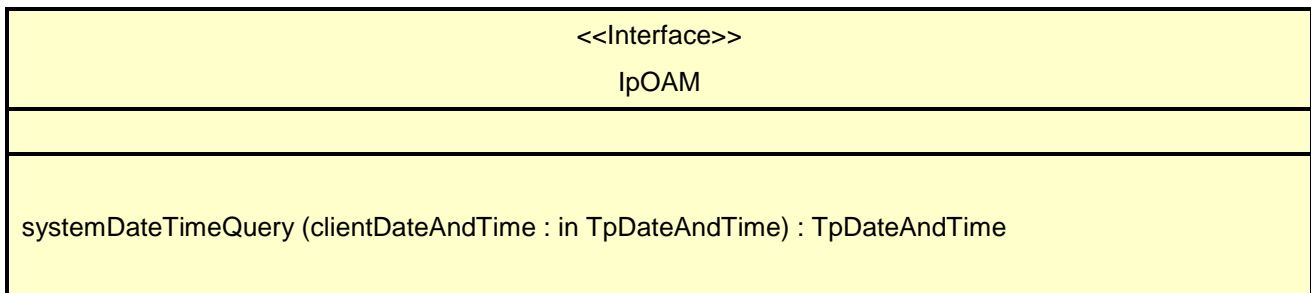
loadStatisticsError : in TpLoadStatisticError

Specifies the error code associated with the failed attempt to retrieve the application's load statistics.

*Raises***TpCommonExceptions****7.3.3.9 Interface Class IpOAM**

Inherits from: IpInterface.

The OAM interface is used to query the system date and time. The application and the framework can synchronise the date and time to a certain extent. Accurate time synchronisation is outside the scope of the OSA APIs. This interface and the systemDateTimeQuery() method are optional.

**7.3.3.9.1 Method systemDateTimeQuery()**

This method is used to query the system date and time. The client application passes in its own date and time to the framework. The framework responds with the system date and time.

Returns <systemDateAndTime> : This is the system date and time of the framework.

Parameters

clientDateAndTime : in TpDateAndTime

This is the date and time of the client (application). The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Returns

TpDateAndTime

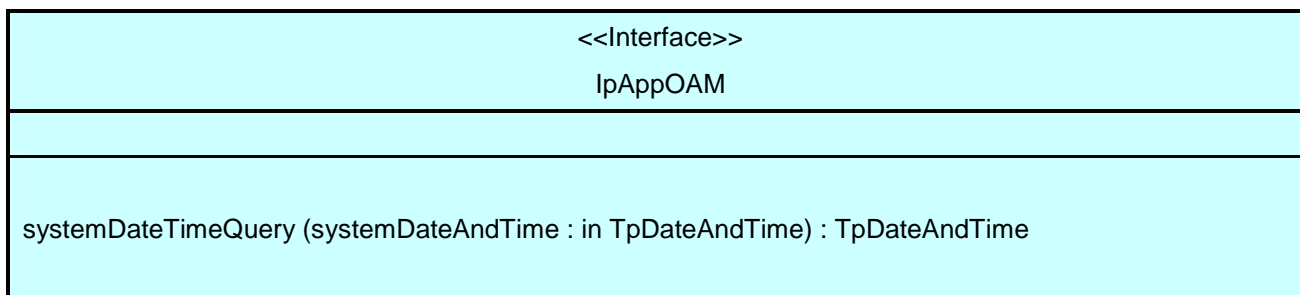
Raises

TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT

7.3.3.10 Interface Class IpAppOAM

Inherits from: IpInterface.

The OAM client application interface is used by the Framework to query the application date and time, for synchronisation purposes. This method is invoked by the Framework to interchange the framework and client application date and time.



7.3.3.10.1 Method systemDateTimeQuery()

This method is used to query the system date and time. The framework passes in its own date and time to the application. The application responds with its own date and time.

Returns <clientDateAndTime> : This is the date and time of the client (application).

Parameters

systemDateAndTime : in TpDateAndTime

This is the system date and time of the framework.

Returns

TpDateAndTime

7.3.4 Event Notification Interface Classes

7.3.4.1 Interface Class IpAppEventNotification

Inherits from: IpInterface.

This interface is used by the framework to inform the application of a generic service-related event. The Event Notification Framework will invoke methods on the Event Notification Application Interface that is specified when the Event Notification interface is obtained.

<<Interface>> IpAppEventNotification
reportNotification (eventInfo : in TpFwEventInfo, assignmentID : in TpAssignmentID) : void notificationTerminated () : void

7.3.4.1.1 Method reportNotification()

This method notifies the application of the arrival of a generic event.

Parameters

eventInfo : in TpFwEventInfo

Specifies specific data associated with this event.

assignmentID : in TpAssignmentID

Specifies the assignment id which was returned by the framework during the createNotification() method. The application can use assignment id to associate events with event specific criteria and to act accordingly.

7.3.4.1.2 Method notificationTerminated()

This method indicates to the application that all generic event notifications have been terminated (for example, due to faults detected).

Parameters

No Parameters were identified for this method.

7.3.4.2 Interface Class IpEventNotification

Inherits from: IpInterface.

The event notification mechanism is used to notify the application of generic service related events that have occurred. If Event Notifications are supported by a Framework, this interface and the createNotification() and destroyNotification() methods shall be supported.

<<Interface>> IpEventNotification
createNotification (eventCriteria : in TpFwEventCriteria) : TpAssignmentID destroyNotification (assignmentID : in TpAssignmentID) : void

7.3.4.2.1 Method createNotification()

This method is used to enable generic notifications so that events can be sent to the application.

Returns <assignmentID> : Specifies the ID assigned by the framework for this newly installed notification.

Parameters

eventCriteria : in **TpFwEventCriteria**

Specifies the event specific criteria used by the application to define the event required.

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

7.3.4.2.2 Method destroyNotification()

This method is used by the application to delete generic notifications from the framework.

Parameters

assignmentID : in **TpAssignmentID**

Specifies the assignment ID given by the framework when the previous createNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENTID.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_ASSIGNMENT_ID

7.4 State Transition Diagrams

This clause contains the State Transition Diagrams for the objects that implement the Framework interfaces on the gateway side. The State Transition Diagrams show the behaviour of these objects. For each state the methods that can be invoked by the application are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the application also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

7.4.1 Service Discovery State Transition Diagrams

7.4.1.1 State Transition Diagrams for IpServiceDiscovery

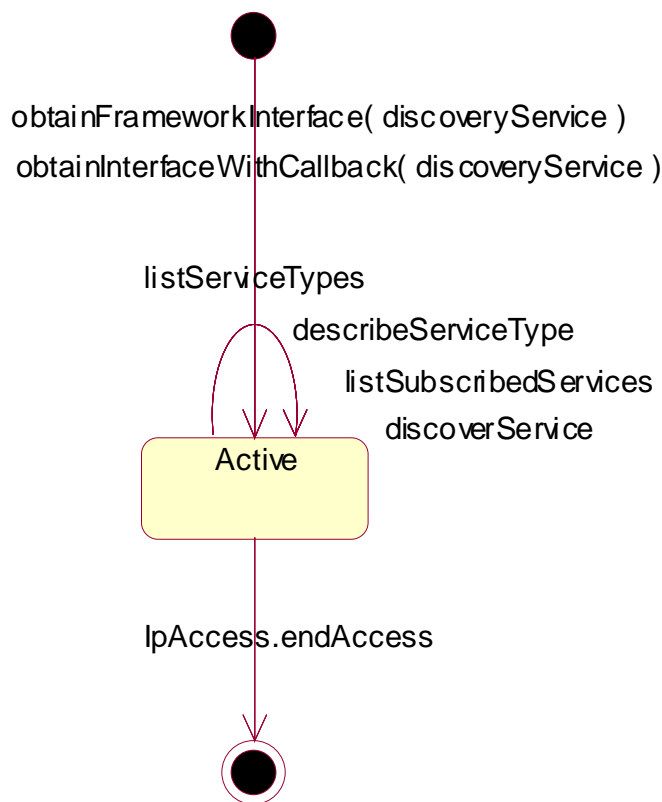


Figure : State Transition Diagram for IpServiceDiscovery

7.4.1.1.1 Active State

When the application requests Service Discovery by invoking the obtainInterface or the obtainInterfaceWithCallback methods on the IpAccess interface, an instance of the IpServiceDiscovery will be created. Next the application is allowed to request a list of the provided SCFs and to obtain a reference to interfaces of SCFs.

7.4.2 Service Agreement Management State Transition Diagrams

There are no State Transition Diagrams defined for Service Agreement Management

7.4.3 Integrity Management State Transition Diagrams

7.4.3.1 State Transition Diagrams for IpLoadManager

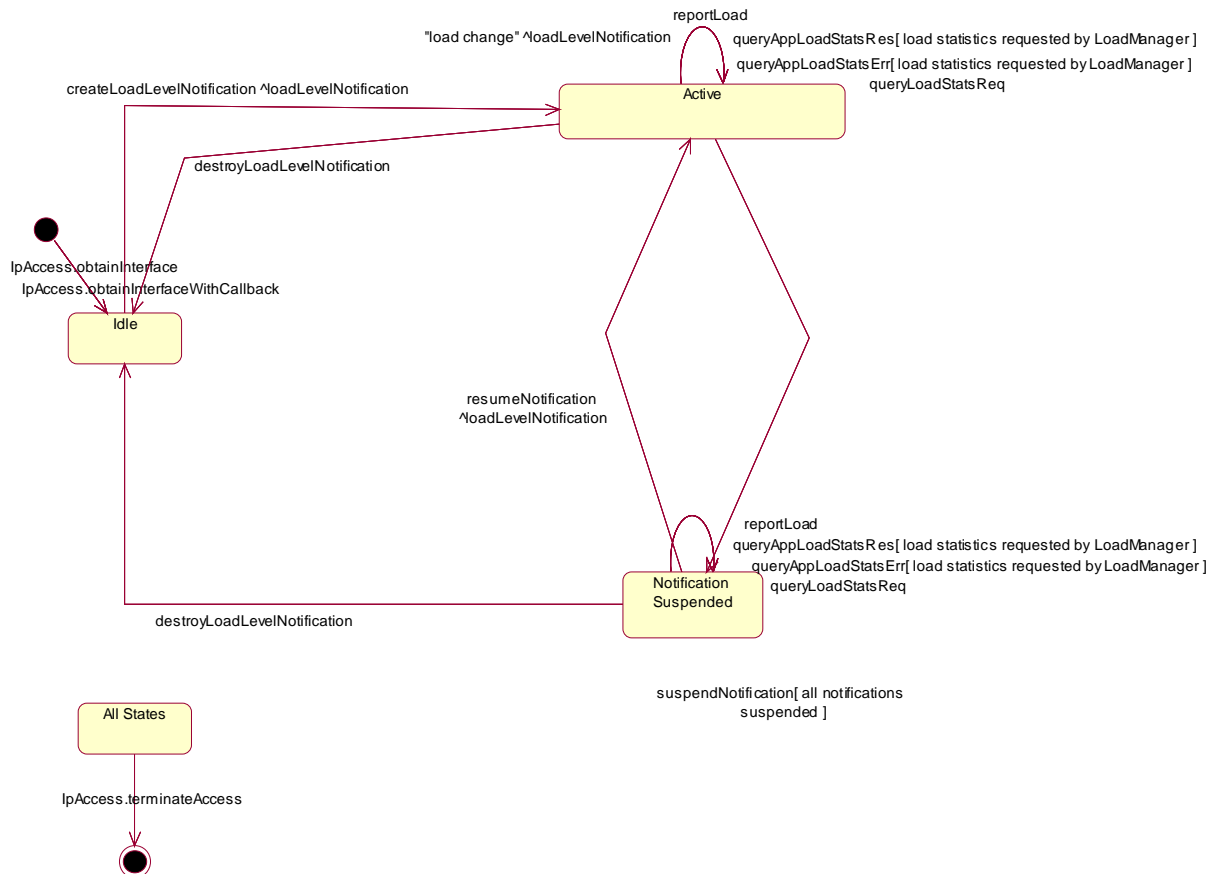


Figure : State Transition Diagram for IpLoadManager

7.4.3.1.1 Idle State

In this state the application has obtained an interface reference of the LoadManager from the IpAccess interface.

7.4.3.1.2 Notification Suspended State

Due to e.g. a temporary load condition, the application has requested the LoadManager to suspend sending the load level notification information.

7.4.3.1.3 Active State

In this state the application has indicated its interest in notifications by performing a createLoadLevelNotification() invocation on the IpLoadManager. The load manager can now request the application to supply load statistics information (by invoking queryAppLoadStatsReq()). Furthermore the LoadManager can request the application to control its load (by invoking loadLevelNotification(), resumeNotification() or suspendNotification() on the application side of interface). In case the application detects a change in load level, it reports this to the LoadManager by calling the method reportLoad().

7.4.3.2 State Transition Diagrams for LoadManagerInternal

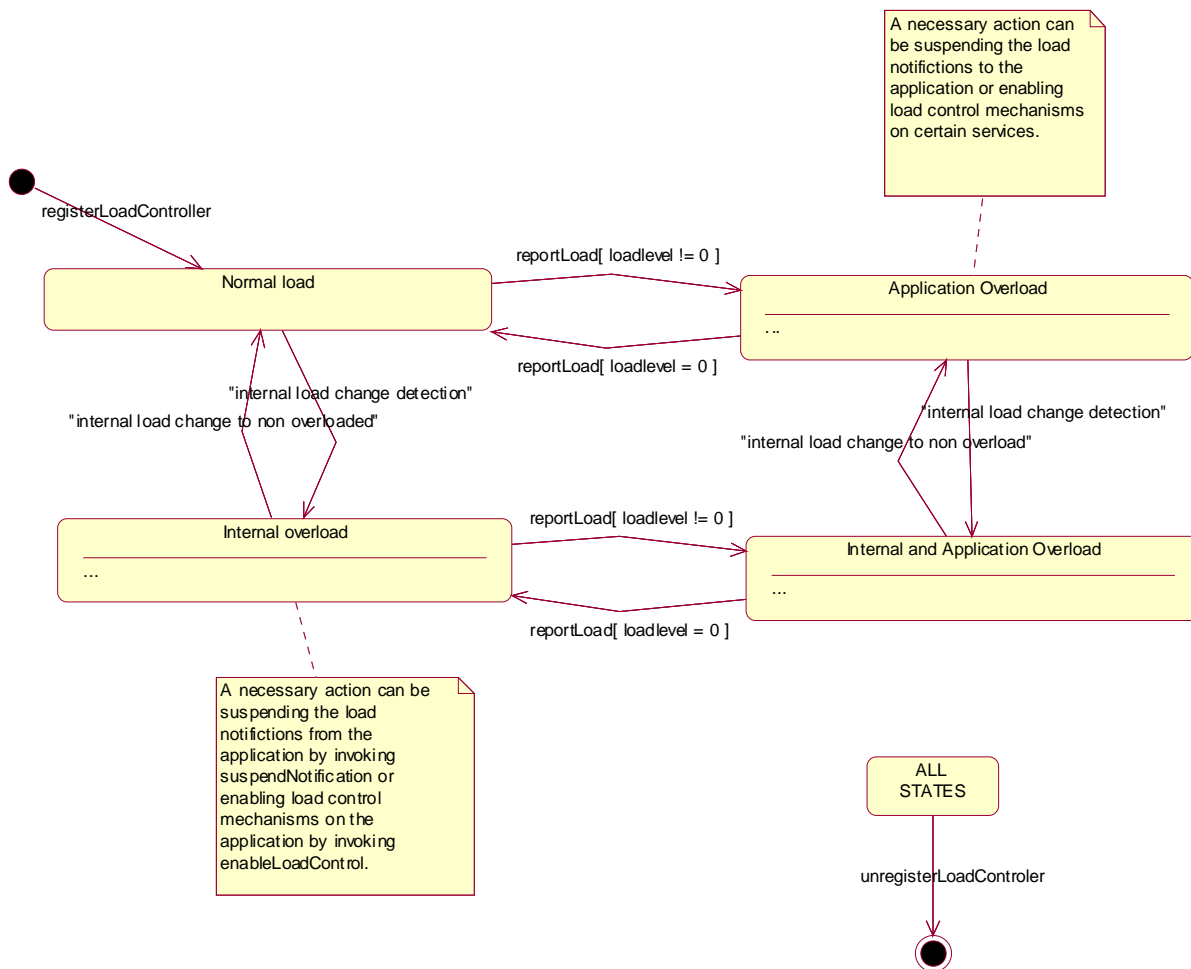


Figure : State Transition Diagram for LoadManagerInternal

7.4.3.2.1 Normal load State

In this state none of the entities defined in the load balancing policy between the application and the framework / SCFs is overloaded.

7.4.3.2.2 Application Overload State

In this state the application has indicated it is overloaded. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

7.4.3.2.3 Internal overload State

In this state the Framework or one or more of the SCFs within the specific load policy is overloaded. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

7.4.3.2.4 Internal and Application Overload State

In this state the application is overloaded as well as the Framework or one or more of the SCFs within the specific load policy. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

7.4.3.3 State Transition Diagrams for IpOAM

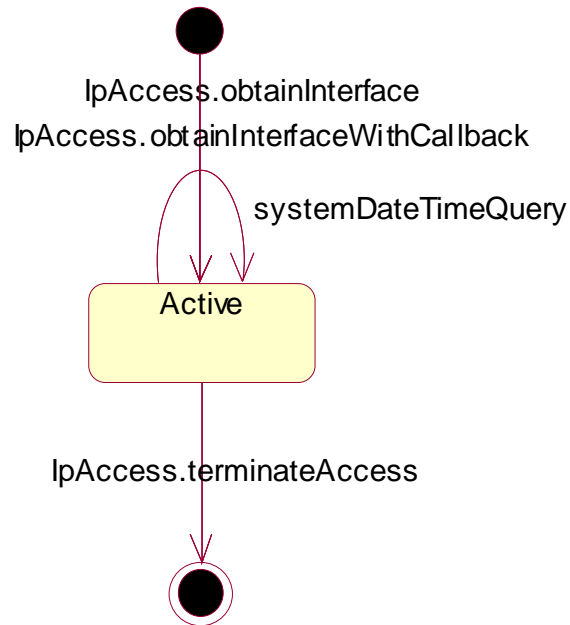


Figure : State Transition Diagram for IpOAM

7.4.3.3.1 Active State

In this state the application has obtained a reference to the IpOAM interface. The application is now able to request the date / time of the Framework.

7.4.3.4 State Transition Diagrams for IpFaultManager

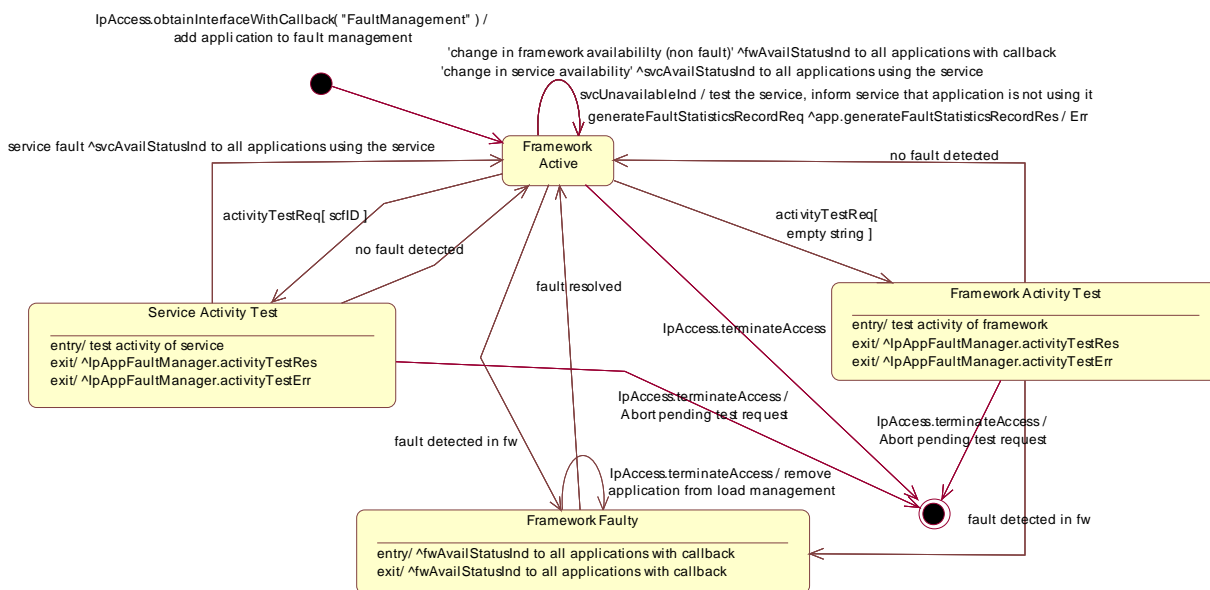


Figure : State Transition Diagram for IpFaultManager

7.4.3.4.1 Framework Active State

This is the normal state of the framework, which is fully functional and able to handle requests from both applications and services capability features.

7.4.3.4.2 Framework Faulty State

In this state, the framework has detected an internal problem with itself such that application and services capability features cannot communicate with it anymore; attempts to invoke any methods that belong to any SCFs of the framework return an error. If the framework ever recovers, applications with fault management callbacks will be notified via a fwAvailStatusInd message.

7.4.3.4.3 Framework Activity Test State

In this state, the framework is performing self-diagnostic test. If a problem is diagnosed, all applications with fault management callbacks are notified through a fwAvailStatusInd message.

7.4.3.4.4 Service Activity Test State

In this state, the framework is performing a test on one service capability feature. If the SCF is faulty, applications with fault management callbacks are notified accordingly through a svcAvailStatusInd message.

7.4.4 Event Notification State Transition Diagrams

7.4.4.1 State Transition Diagrams for IpEventNotification

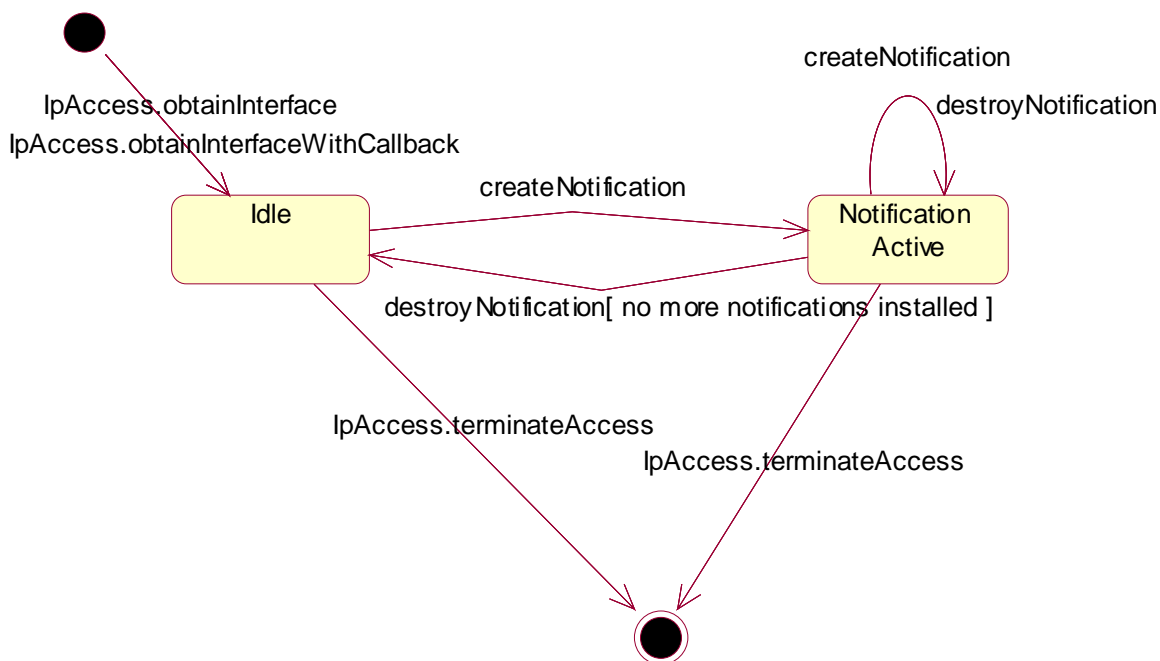


Figure : State Transition Diagram for IpEventNotification

8 Framework-to-Service API

8.1 Sequence Diagrams

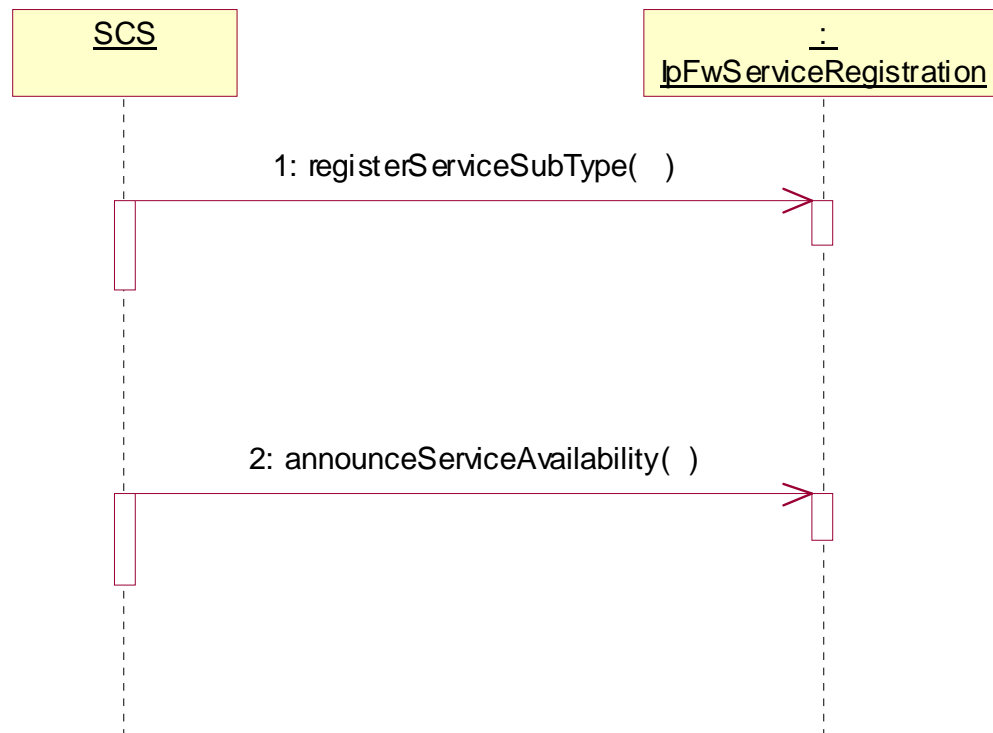
8.1.1 Service Discovery Sequence Diagrams

No Sequence Diagrams exist for Service Discovery

8.1.2 Service Registration Sequence Diagrams

8.1.2.1 New SCF Sub Type Registration

The following figure shows the process of registering a new proprietary Service Capability Feature in the Framework. This SCF is a sub type of the standard SCF.

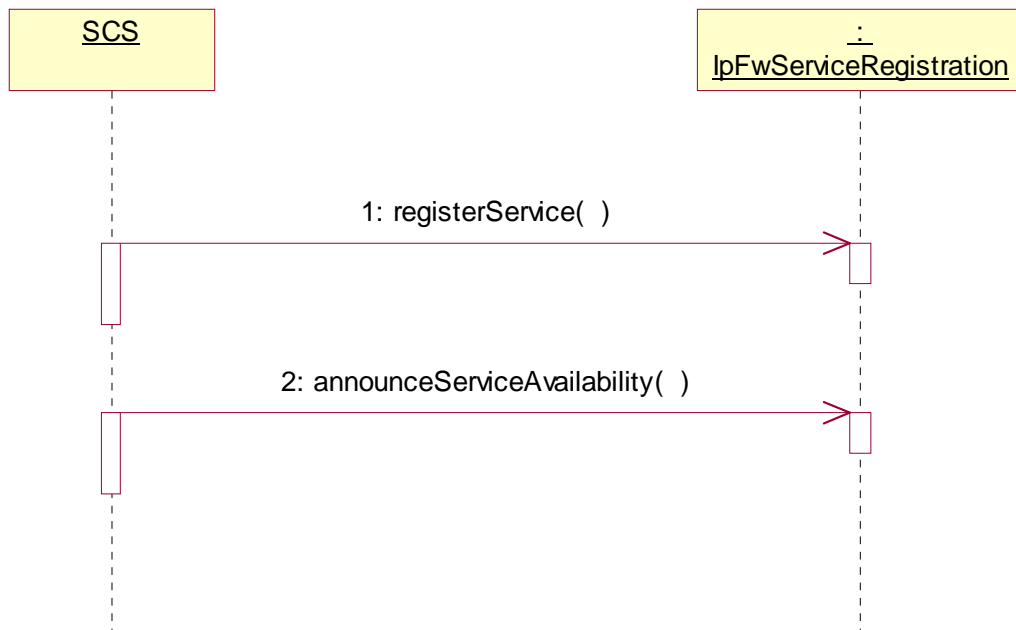


1: Registration: first step - register service sub type. For sub type registration, besides the values for the standard service properties, the modes, types, and values for the additional service properties must be provided by the SCF.

2: Registration: second step - announce service availability. This is identical to announcing availability of super types.

8.1.2.2 New SCF Registration

The following figure shows the process of registering a new Service Capability Feature in the Framework. Service Registration is a two step process:



1: Registration: first step - register service.

The purpose of this first step in the process of registration is to agree, within the network, on a name to call, internally, a newly installed SCF version. It is necessary because the OSA Framework and SCF in the same network may come from different vendors. The goal is to make an association between the new SCF version, as characterized by a list of properties, and an identifier called serviceID.

This service ID will be the name used in that network (that is, between that network's Framework and its SCSs), whenever it is necessary to refer to this newly installed version of SCF (for example for announcing its availability, or for withdrawing it later).

The following input parameters are given from the SCS to the Framework in this first registration step:

- in serviceName

This is a string with the name of the SCF, among a list of standard names (e.g. "P_MPCC").

- in servicePropertyList

This is a list of types TpServiceProperty; each TpServiceProperty is a pair of (ServicePropertyName, ServicePropertyValueList).

- ServicePropertyName is a string that defines a valid SCF property name (valid SCF property names are listed in the SCF data definition).
- ServicePropertyValueList is a numbered set of types TpServicePropertyValue; TpServicePropertyValue is a string that describes a valid value of a SCF property (valid SCF property values are listed in the SCF data definition).

The following output parameter results from service registration:

- out serviceID

This is a string, automatically generated by the Framework and unique within the Framework.

This is the name by which the newly installed version of SCF, described by the list of properties above, is going to be identified internally in this network.

2: Registration: second step - announce service availability.

At this point the network's Framework is aware of the existence of a new SCF, and could let applications know - but they would have no way to use it. Installing the SCS logic and assigning a name to it does not make this SCF available. In order to make the SCF available an "entry point", called lifecycle manager, is used. The role of the lifecycle manager is to control the life cycle of an interface, or set of interfaces, and provide clients with the references that are necessary to invoke the methods offered by these interfaces. The starting point for a client to use an SCF is to obtain an interface reference to a lifecycle manager of the desired SCF.

A Network Operator, upon completion of the first registration phase, and once it has an identifier to the new SCF version, will instantiate a lifecycle manager for it that will allow client to use it. Then it will inform the Framework of the value of the interface associated to the new SCF. After the receipt of this information, the Framework makes the new SCF (identified by the pair [serviceID, serviceInstanceLifecycleManagerRef]) discoverable.

The following input parameters are given from the SCS to the Framework in this second registration step:

- in serviceID.

This is the identifier that has been agreed in the network for the new SCF; any interaction related to the SCF needs to include the serviceID, to know which SCF it is.

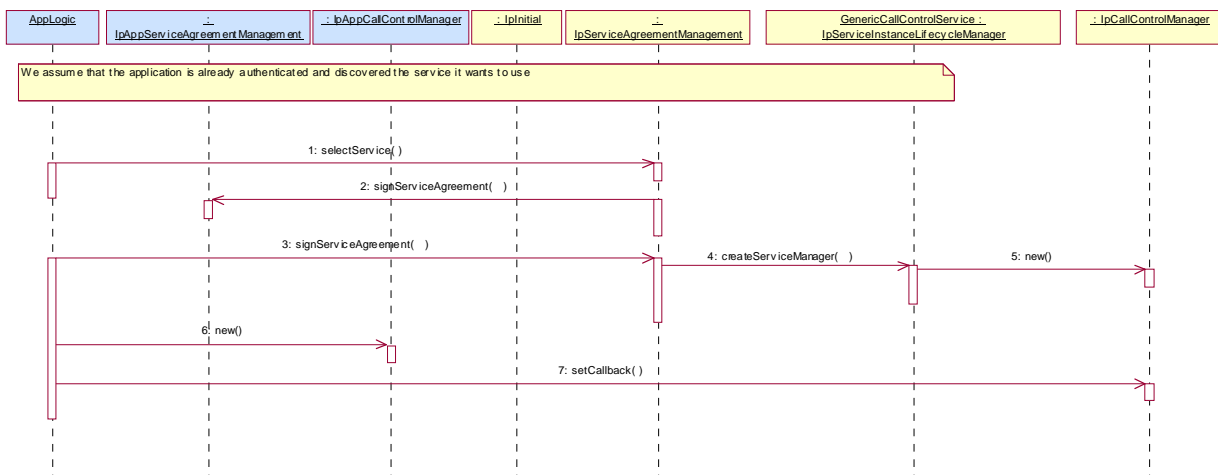
- in serviceInstanceLifecycleManagerRef.

This is the interface reference at which the lifecycle manager of the new SCF is available. Note that the Framework will have to invoke the method createServiceManager() in this interface when a client application signs an agreement to use the SCF so that it can get the service manager interface necessary for applications as an entry point to any SCF.

8.1.3 Service Instance Lifecycle Manager Sequence Diagrams

8.1.3.1 Sign Service Agreement

This sequence illustrates how the application can get access to a specified service. It only illustrates the last part: the signing of the service agreement and the corresponding actions towards the service. For more information on accessing the framework, authentication and discovery of services, see the corresponding clauses.



1: The application selects the service, using a serviceID for the generic call control service. The serviceID could have been obtained via the discovery interface. A ServiceToken is returned to the application.

2: The client application signs the service agreement.

3: The framework signs the service agreement. As a result a service manager interface reference (in this case of type IpCallControlManager) is returned to the application.

4: Provided the signature information is correct and all conditions have been fulfilled, the framework will request the service identified by the serviceID to return a service manager interface reference. The service manager is the initial point of contact to the service.

5: The lifecycle manager creates a new manager interface instance (a call control manager) for the specified application. It should be noted that this is an implementation detail. The service implementation may use other mechanism to get a service manager interface instance.

Following the creation of the service manager outlined above, a unique instance of the service particular to the application client results. This service instance is assigned a serviceInstanceID by the Framework, which is provided to the Service Instance Lifecycle manager during the createServiceManager operation. If it is necessary that Framework Integrity Management functionality and operations are to be supported between the Framework and the service instance identified by the defined serviceInstanceID, it is then necessary for the new service instance to establish an access session with the Framework. This provides the Framework with the ability to manage and monitor the operation of the service instance that relates to a particular application client. The steps required to establish a Framework access session are outlined in clause 6 of the present document.

6: The application creates a new IpAppCallControlManager interface to be used for callbacks.

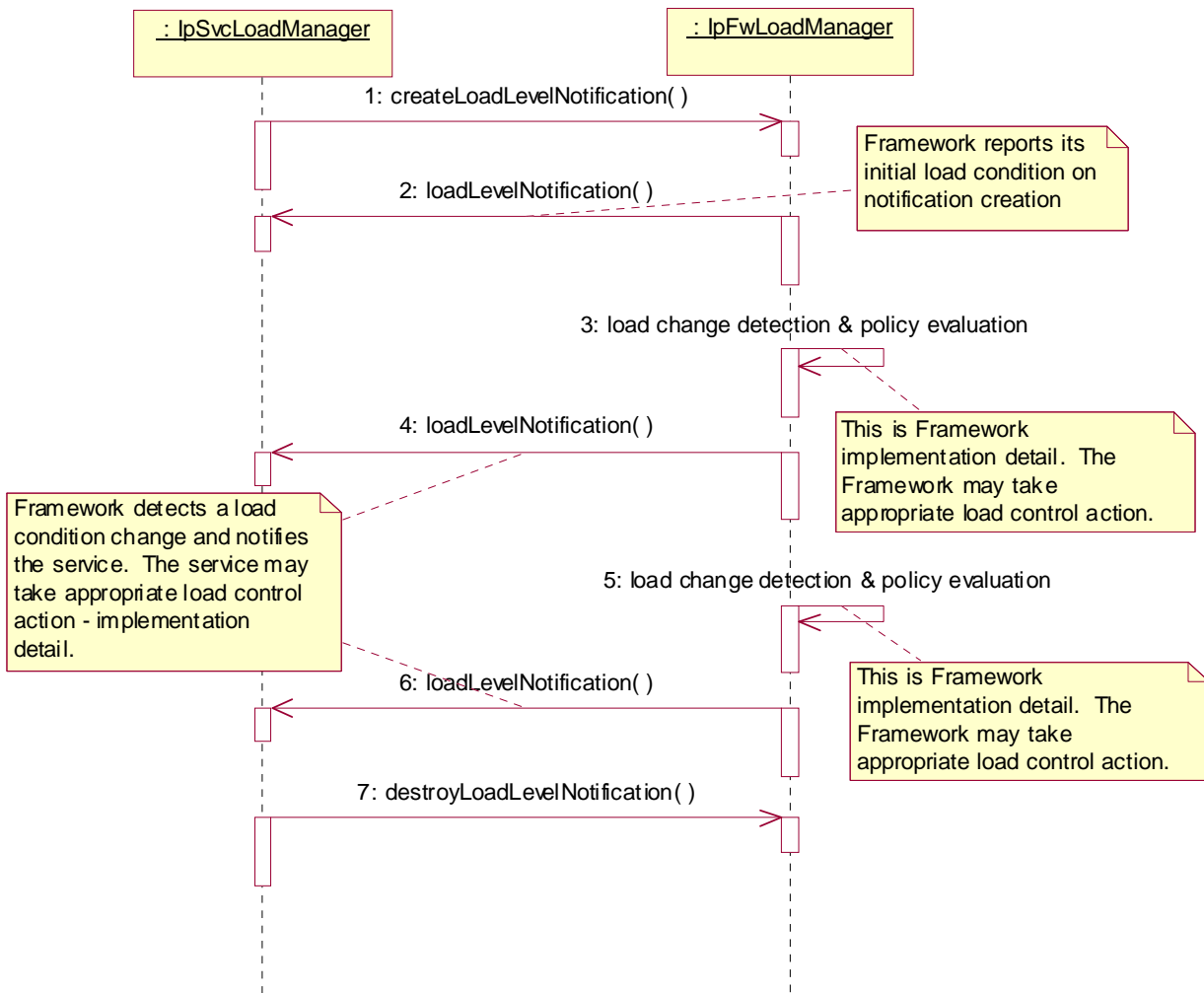
7: The Application sets the callback interface to the interface created with the previous message.

An application (identifiable by a given TpClientAppID may carry out the sequence, as exemplified above, multiple times.

8.1.4 Integrity Management Sequence Diagrams

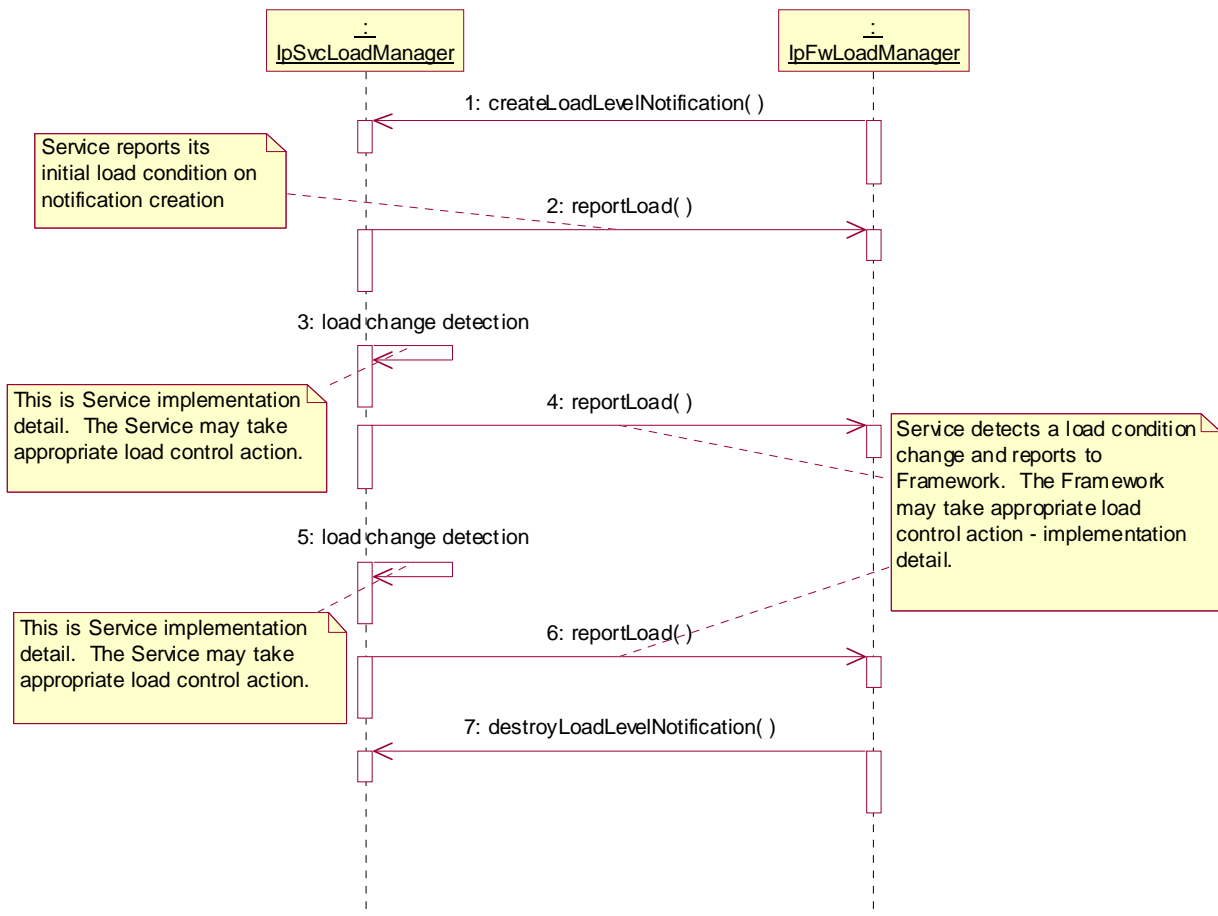
8.1.4.1 Load Management: Service callback registration and load control

This sequence diagram shows how a service registers itself and the framework invokes load management function based on policy.

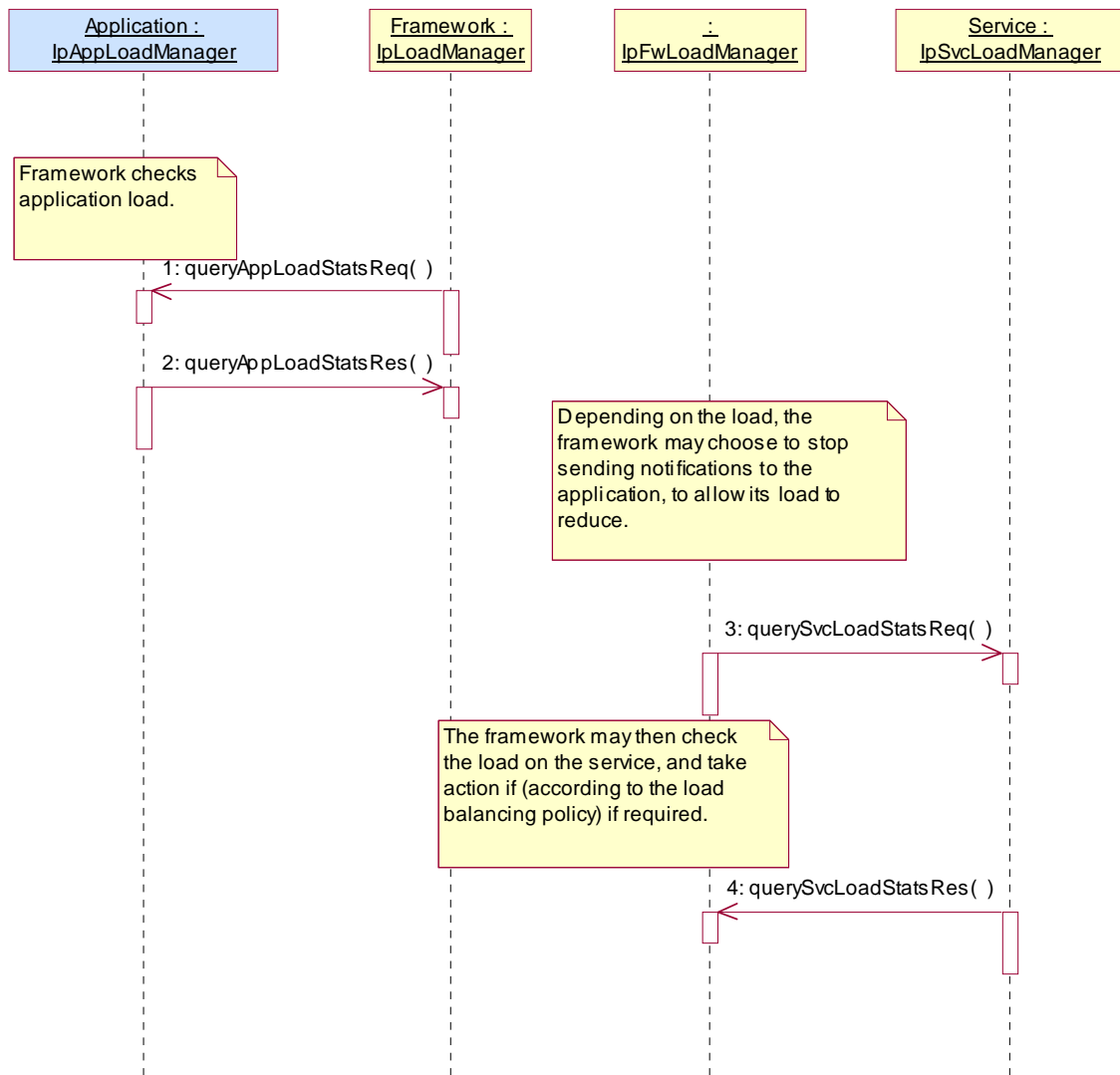


8.1.4.2 Load Management: Framework callback registration and service load control

This sequence diagram shows how the framework registers itself and the service invokes load management function to inform the framework of service load.

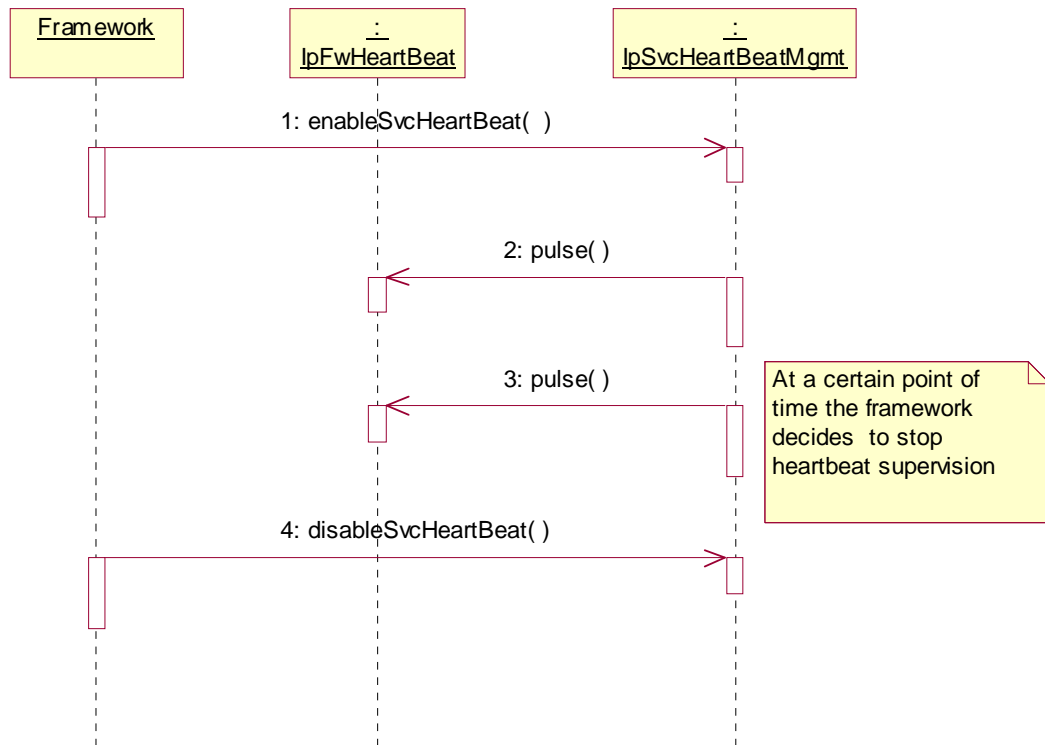


8.1.4.3 Load Management: Client and Service Load Balancing

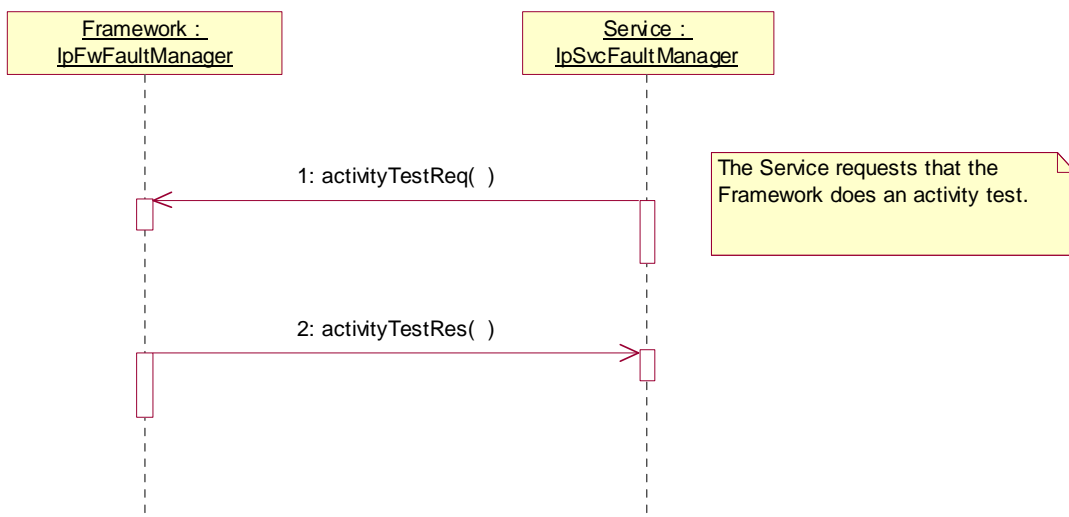


8.1.4.4 Heartbeat Management: Start/perform/end heartbeat supervision of the service

In this sequence diagram, the framework has decided that it wishes to monitor the service, and has therefore requested the service to commence sending its heartbeat. The service responds by sending its heartbeat at the specified interval. The framework then decides that it is satisfied with the service's health and disables the heartbeat mechanism. If the heartbeat was not received from the service within the specified interval, the framework can decide that the service has failed the heartbeat and can then perform some recovery action.



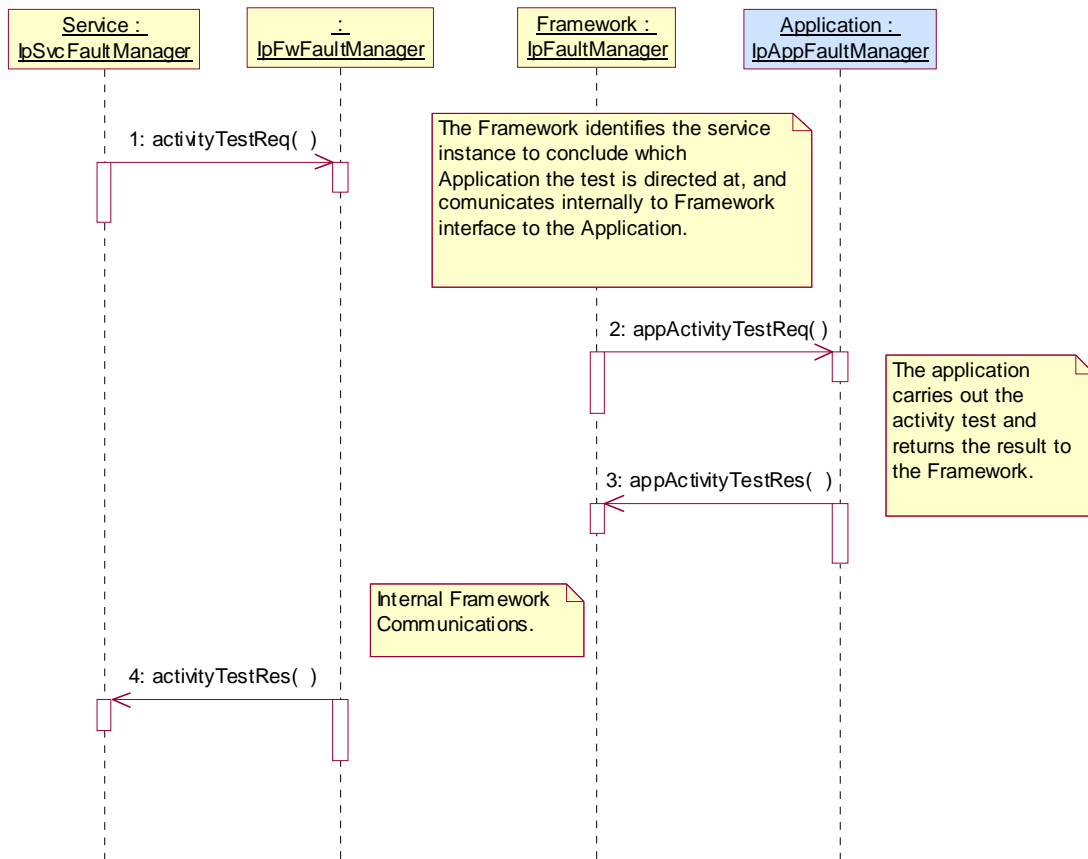
8.1.4.5 Fault Management: Service requests Framework activity test



1: The service asks the framework to carry out its activity test. The service denotes that it requires the activity test done for the framework, rather than an application, by supplying an appropriate parameter.

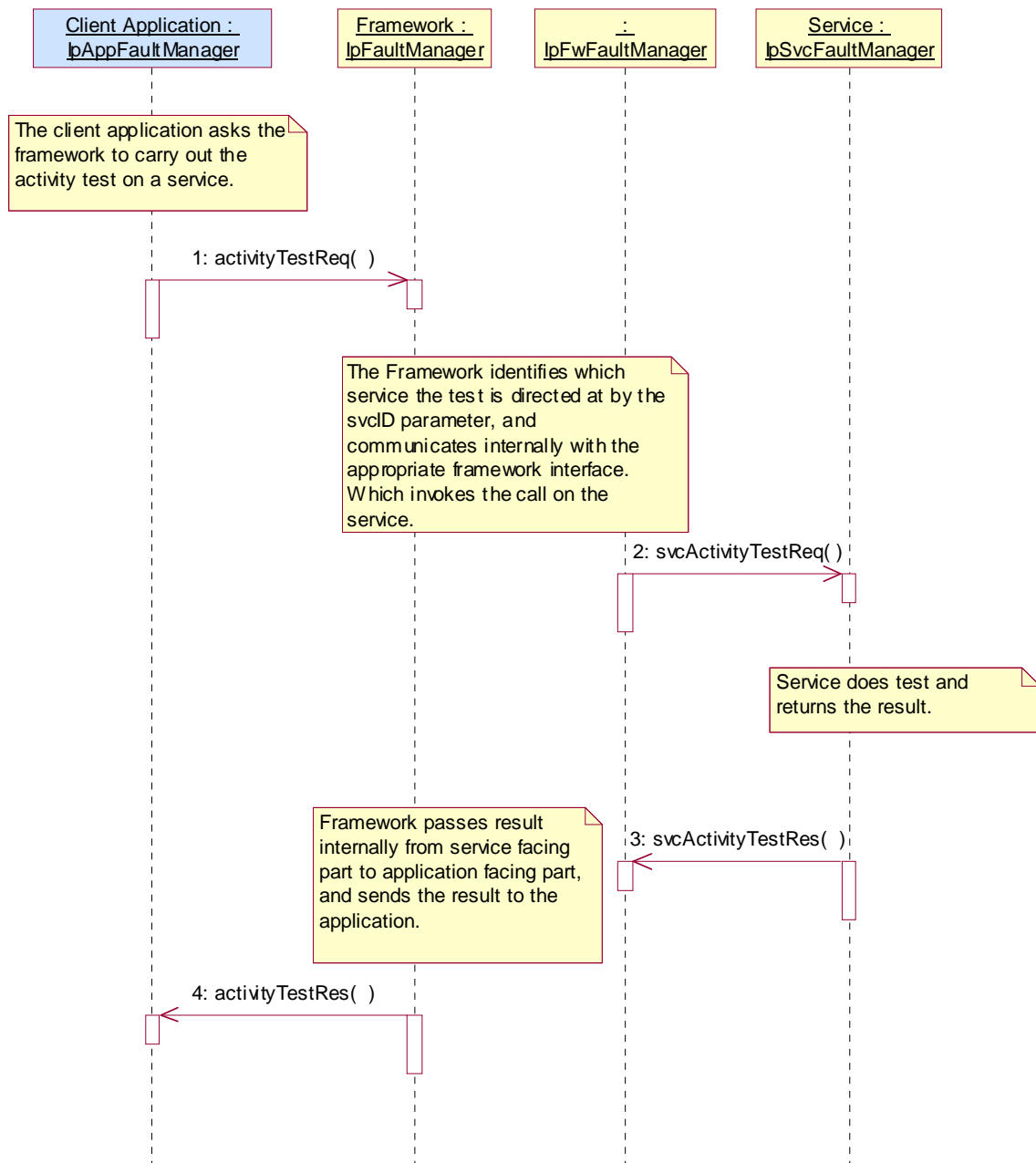
2: The framework carries out the test and returns the result to the service.

8.1.4.6 Fault Management: Service requests Application activity test



- 1: The service instance asks the framework to invoke an activity test on the client application.
- 2: The framework asks the application to do the activity test. It is assumed that there is internal communication between the service facing part of the framework (i.e. IpFwFaultManager interface) and the part that faces the client application.
- 3: The application does the activity test and returns the result to the framework.
- 4: The framework internally passes the result from its application facing interface (IpFaultManager) to its service facing side, and sends the result to the service.

8.1.4.7 Fault Management: Application requests Service activity test



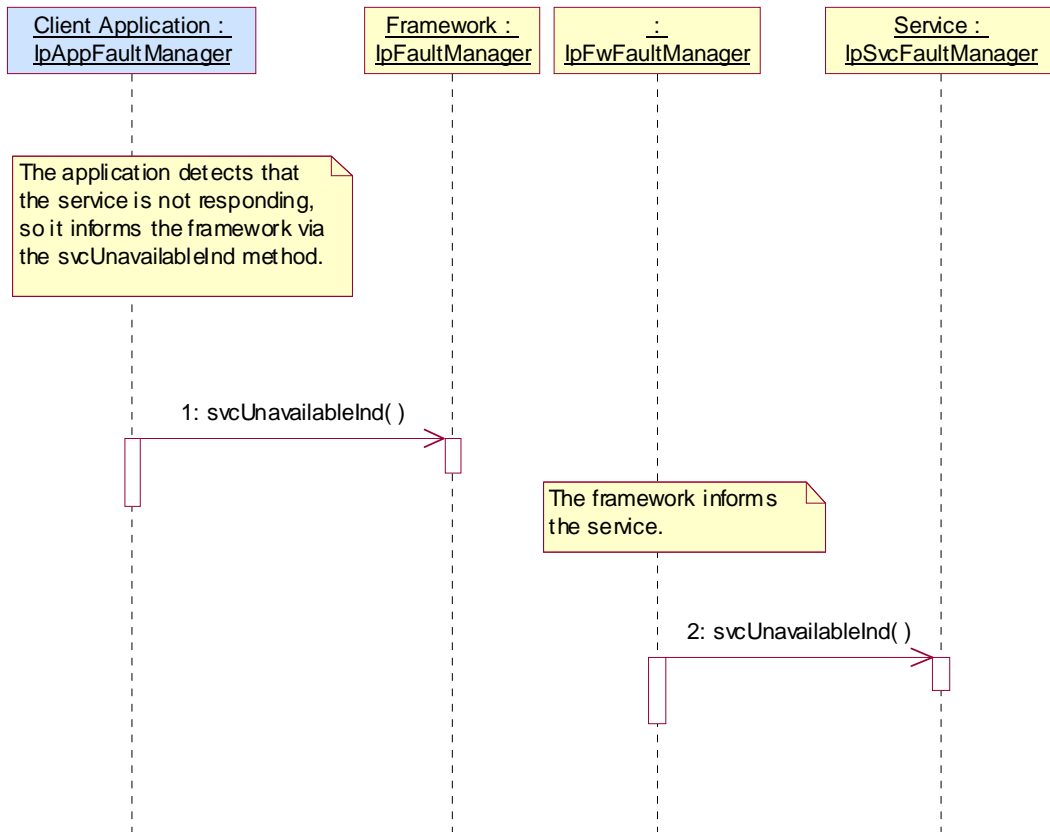
1: The client application asks the framework to invoke an activity test on a service, the service is identified by the svcId parameter.

2: The framework asks the service to do the activity test. It is assumed that there is internal communication between the application facing part of the framework (i.e. IpFaultManager interface) and the part that faces the service.

3: The service does the activity test and returns the result to the framework.

4: The framework internally passes the result from its service facing interface (IpFwFaultManager) to its application facing side, and sends the result to the client application.

8.1.4.8 Fault Management: Application detects service is unavailable



1: The client application detects that the service instance is currently not available, i.e. the service instance is not responding to the client application in the normal way, so it informs the framework.

2: The framework informs the service instance that the client application was unable to get a response from it and can no longer use the service instance. The service or framework may then decide to carry out an activity test to see whether there is a general problem with the service instance that requires further action.

8.1.5 Event Notification Sequence Diagrams

No Sequence Diagrams exist for Event Notification

8.2 Class Diagrams

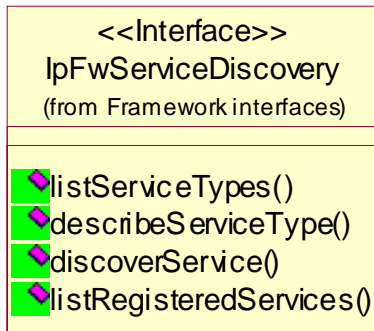


Figure: Service Discovery Package Overview

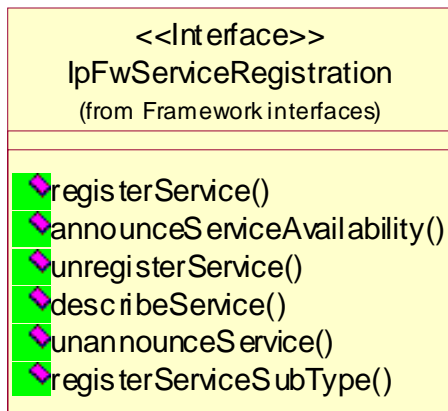


Figure: Service Registration Package Overview



Figure: Trust and Security Management Package Overview

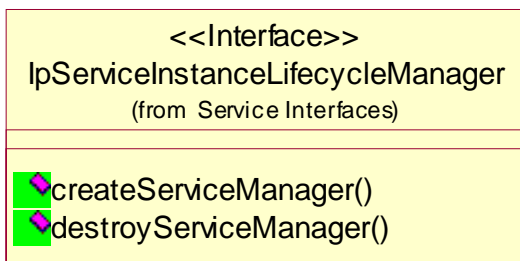


Figure: Service Instance Lifecycle Manager Package Overview

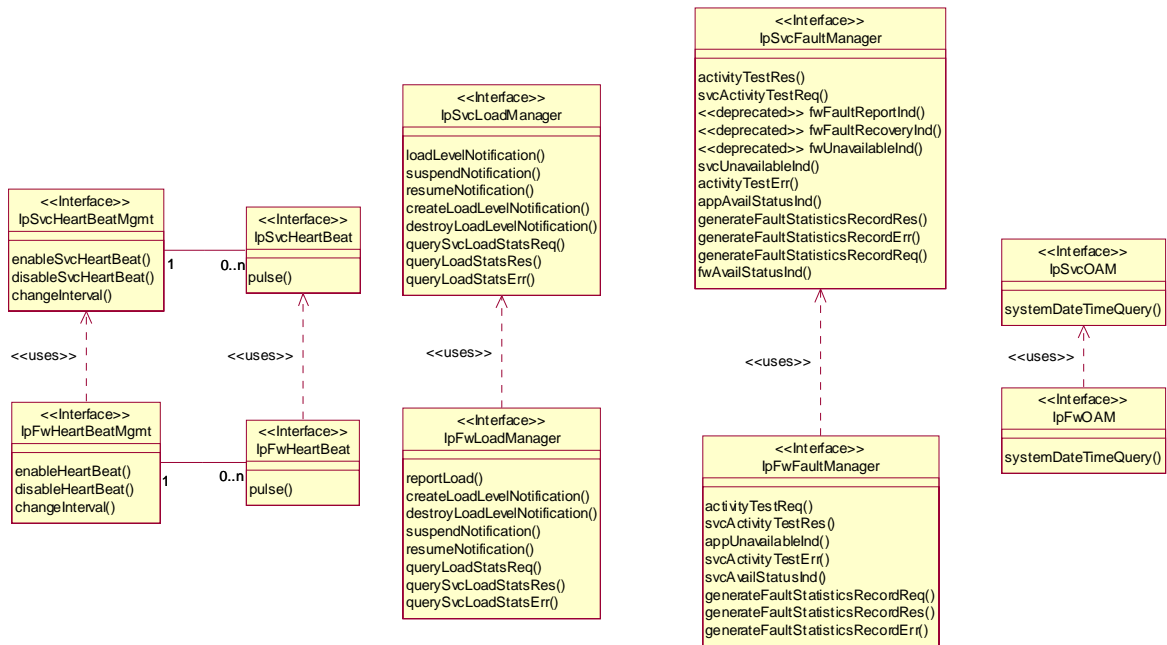


Figure: Integrity Management Package Overview

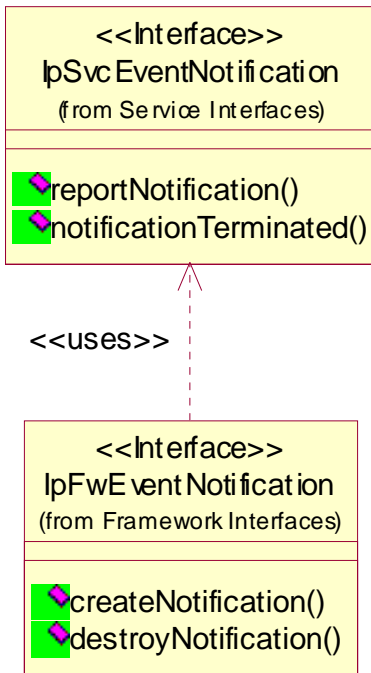


Figure: Event Notification Package Overview

8.3 Interface Classes

8.3.1 Service Registration Interface Classes

Before a service can be brokered (discovered, subscribed, accessed, etc.) by an enterprise, it has to be registered with the Framework. Services are registered against a particular service type. Therefore service types are created first, and then services corresponding to those types are accepted from the Service Suppliers for registration in the framework. The framework maintains a repository of service types and registered services.

In order to register a new service in the framework, the service supplier must select a service type and the "property values" for the service. The service discovery functionality described in the previous clause enables the service supplier to obtain a list of all the service types supported by the framework and their associated sets of service property values.

The Framework service registration-related interfaces are invoked by third party service supplier's administrative applications. They are described below. Note that these methods cannot be invoked until the authentication methods have been invoked successfully.

8.3.1.1 Interface Class IpFwServiceRegistration

Inherits from: IpInterface.

The Service Registration interface provides the methods used for the registration of network SCFs at the framework. This interface and at least the methods registerService(), announceServiceAvailability(), unregisterService() and unannounceService() shall be implemented by a Framework.

<<Interface>> IpFwServiceRegistration
registerService (serviceTypeName : in TpServiceTypeName, servicePropertyList : in TpServicePropertyList) : TpServiceID announceServiceAvailability (serviceID : in TpServiceID, serviceInstanceLifecycleManagerRef : in service_lifecycle::IpServiceInstanceLifecycleManagerRef) : void unregisterService (serviceID : in TpServiceID) : void describeService (serviceID : in TpServiceID) : TpServiceDescription unannounceService (serviceID : in TpServiceID) : void registerServiceSubType (serviceTypeName : in TpServiceTypeName, servicePropertyList : in TpServicePropertyList, extendedServicePropertyList : in TpServiceTypePropertyValueList) : TpServiceID

8.3.1.1.1 Method registerService()

The registerService() operation is the means by which a service is registered in the Framework, for subsequent discovery by the enterprise applications. Registration can only succeed when the Service type of the service is known to the Framework (ServiceType is 'available'). A service-ID is returned to the service supplier when a service is registered in the Framework. When the service is not registered because the ServiceType is 'unavailable', a P_SERVICE_TYPE_UNAVAILABLE is raised. The service-ID is the handle with which the service supplier can identify the registered service when needed (e.g. for withdrawing it). The service-ID is only meaningful in the context of the Framework that generated it.

This method should be used for registration of service super types only. For registering service sub types, the registerServiceSubType() method should be used.

Returns <serviceID> : This is the unique handle that is returned as a result of the successful completion of this operation. The Service Supplier can identify the registered service when attempting to access it via other operations such as unregisterService(), etc. Enterprise client applications are also returned this service-ID when attempting to discover a service of this type.

If a service is registered with the property P_COMPATIBLE_WITH_SERVICE in its servicePropertyList, then the Framework shall notify all applications using instances of services identified by this property, using the P_EVENT_FW_MIGRATION_SERVICE_AVAILABLE event, if they have registered for such a notification. If an incorrect combination of properties is included in conjunction with P_COMPATIBLE_WITH_SERVICE, a P_MISSING_MANDATORY_PROPERTY exception is raised.

Parameters

serviceTypeName : in TpServiceTypeName

The "serviceTypeName" parameter identifies the service type. If the string representation of the "type" does not obey the rules for identifiers, then a P_ILLEGAL_SERVICE_TYPE exception is raised. If the "type" is correct syntactically but the Framework is able to unambiguously determine that it is not a recognised service type, then a P_UNKNOWN_SERVICE_TYPE exception is raised.

servicePropertyList : in TpServicePropertyList

The "servicePropertyList" parameter is a list of property name and property value pairs. They describe the service being registered. This description typically covers behavioural, non-functional and non-computational aspects of the service. Service properties are marked "mandatory" or "readonly". These property mode attributes have the following semantics:

a. mandatory - a service associated with this service type must provide an appropriate value for this property when registering.

b. readonly - this modifier indicates that the property is optional, but that once given a value, subsequently it may not be modified.

Specifying both modifiers indicates that a value must be provided and that subsequently it may not be modified.

Examples of such properties are those which form part of a service agreement and hence cannot be modified by service suppliers during the life time of service.

If the type or the semantics of the type of any of the property values is not the same as the declared type (declared in the service type), then a P_PROPERTY_TYPE_MISMATCH exception is raised. If the "servicePropertyList" parameter omits any property declared in the service type with a mode of mandatory, then a P_MISSING_MANDATORY_PROPERTY exception is raised. If two or more properties with the same property name are included in this parameter, the P_DUPLICATE_PROPERTY_NAME exception is raised.

Returns

TpServiceID

Raises

TpCommonExceptions, P_PROPERTY_TYPE_MISMATCH, P_DUPLICATE_PROPERTY_NAME, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE, P_MISSING_MANDATORY_PROPERTY, P_SERVICE_TYPE_UNAVAILABLE

8.3.1.1.2 Method announceServiceAvailability()

The registerService() method described previously does not make the service discoverable. The announceServiceAvailability() method is invoked after the service is authenticated and its service instance lifecycle manager is instantiated at a particular interface. This method informs the framework of the availability of "service instance lifecycle manager" of the previously registered service, identified by its service ID, at a specific interface. After the receipt of this method, the framework makes the corresponding service discoverable.

There exists a "service manager" instance per service instance. Each service implements the `IpServiceInstanceLifecycleManager` interface. The `IpServiceInstanceLifecycleManager` interface supports a method called the `createServiceManager(application: in TpClientAppID, serviceProperties : in TpServicePropertyList, serviceInstanceID : in TpServiceInstanceID) : IpServiceRef`. When the service agreement is signed for some serviceID (using `signServiceAgreement()`), the framework calls the `createServiceManager()` for this service, gets a `serviceManager` and returns this to the client application.

Parameters

serviceID : in TpServiceID

The service ID of the service that is being announced. If the string representation of the "serviceID" does not obey the rules for service identifiers, then a `P_ILLEGAL_SERVICE_ID` exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then a `P_UNKNOWN_SERVICE_ID` exception is raised.

**serviceInstanceLifecycleManagerRef : in
service_lifecycle : IpServiceInstanceLifecycleManagerRef**

The interface reference at which the service instance lifecycle manager of the previously registered service is available.

Raises

**TpCommonExceptions, P_ILLEGAL_SERVICE_ID, P_UNKNOWN_SERVICE_ID,
P_INVALID_INTERFACE_TYPE**

8.3.1.1.3 Method `unregisterService()`

The `unregisterService()` operation is used by the service suppliers to remove a registered service from the Framework. The service is identified by the "service-ID" which was originally returned by the Framework in response to the `registerService()` operation. The service must be in the SCF Registered state. All instances of the service will be deleted.

Parameters

serviceID : in TpServiceID

The service to be withdrawn is identified by the "serviceID" parameter which was originally returned by the `registerService()` operation. If the string representation of the "serviceID" does not obey the rules for service identifiers, then a `P_ILLEGAL_SERVICE_ID` exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then a `P_UNKNOWN_SERVICE_ID` exception is raised.

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_ID, P_UNKNOWN_SERVICE_ID

8.3.1.1.4 Method `describeService()`

The `describeService()` operation returns the information about a service that is registered in the framework. It comprises, the "type" of the service, and the "properties" that describe this service. The service is identified by the "service-ID" parameter which was originally returned by the `registerService()` operation.

The SCF may register various versions of the same SCF, each with a different description (more or less restrictive, for example), and each getting a different serviceID assigned.

Returns <serviceDescription> : This consists of the information about an offered service that is held by the Framework. It comprises the "type" of the service , and the properties that describe this service.

Parameters

serviceID : in TpServiceID

The service to be described is identified by the "serviceID" parameter which was originally returned by the registerService() operation. If the string representation of the "serviceID" does not obey the rules for object identifiers, then an P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then a P_UNKNOWN_SERVICE_ID exception is raised.

Returns

TpServiceDescription

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_ID, P_UNKNOWN_SERVICE_ID

8.3.1.1.5 Method unannounceService()

This method results in the service no longer being discoverable by applications. It is, however, still registered and the service ID is still associated with it. Applications currently using the service can continue to use the service but no new applications should be able to start using the service. Also, all unused service tokens relating to the service will be expired. This will prevent anyone who has already performed a selectService() but not yet performed the signServiceAgreement() from being able to obtain a new instance of the service.

Parameters

serviceID : in TpServiceID

The service ID of the service that is being unannounced. If the string representation of the "serviceID" does not obey the rules for service identifiers, then a P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then a P_UNKNOWN_SERVICE_ID exception is raised.

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_ID, P_UNKNOWN_SERVICE_ID

8.3.1.1.6 Method registerServiceSubType()

The registerServiceSubType() operation is the means by which an extended service is registered in the Framework, for subsequent discovery by the enterprise applications. Registration only succeeds if the service type is known to the Framework (ServiceType is 'available'). A service-ID is returned to the service supplier when a service is registered in the Framework. When the service is not registered because the ServiceType is 'unavailable', a P_SERVICE_TYPE_UNAVAILABLE exception is raised. The service-ID is the handle with which the service supplier can identify the registered service when needed (e.g. for withdrawing it). The service-ID is only meaningful in the context of the Framework that generated it.

This method should be used for registration of service sub types only. For registering service super types, the registerService () method should be used.

Returns <serviceID> : This is the unique handle that is returned as a result of the successful completion of this operation. The Service Supplier can identify the registered service when attempting to access it via other operations

such as `unregisterService()`, etc. Enterprise client applications are also returned this service-ID when attempting to discover a service of this type.

Parameters

serviceName : in TpServiceName

The "serviceName" parameter identifies the service type. If the string representation of the "type" does not obey the rules for identifiers, then a `P_ILLEGAL_SERVICE_TYPE` exception is raised. If the "type" is correct syntactically but the Framework is able to unambiguously determine that it is not a recognised service type, then a `P_UNKNOWN_SERVICE_TYPE` exception is raised.

servicePropertyList : in TpServicePropertyList

The "servicePropertyList" parameter is a list of property name and property value pairs corresponding to the service properties applicable to the standard service. They describe the service being registered.

If the type or the semantics of the type of any of the property values is not the same as the declared type (declared in the service type), then a `P_PROPERTY_TYPE_MISMATCH` exception is raised.

If the "servicePropertyList" parameter omits any property declared in the service type with a mode of mandatory, then a `P_MISSING_MANDATORY_PROPERTY` exception is raised.

If two or more properties with the same property name are included in this parameter, the `P_DUPLICATE_PROPERTY_NAME` exception is raised.

extendedServicePropertyList : in TpServiceTypePropertyValueList

The "extendedServicePropertyList" parameter is a list of property name, mode, type, and property value tuples corresponding to the service properties applicable to the extended standard service. They describe the service being registered.

If two or more properties with the same property name are included in this parameter, the `P_DUPLICATE_PROPERTY_NAME` exception is raised.

Returns

TpServiceID

Raises

TpCommonExceptions, P_PROPERTY_TYPE_MISMATCH, P_DUPLICATE_PROPERTY_NAME, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE, P_MISSING_MANDATORY_PROPERTY, P_SERVICE_TYPE_UNAVAILABLE

8.3.2 Service Instance Lifecycle Manager Interface Classes

The `IpServiceInstanceLifecycleManager` interface allows the framework to get access to a service manager interface of a service. It is used during the `signServiceAgreement`, in order to return a service manager interface reference to the application. Each service has a service manager interface that is the initial point of contact for the service. E.g. the generic call control service uses the `IpCallControlManager` interface.

8.3.2.1 Interface Class `IpServiceInstanceLifecycleManager`

Inherits from: `IpInterface`.

The `IpServiceInstanceLifecycleManager` interface allows the Framework to create and destroy Service Manager Instances. This interface and the `createServiceManager()` and `destroyServiceManager()` methods shall be implemented by a Service.

<<Interface>> IpServiceInstanceLifecycleManager
<pre> createServiceManager (application : in TpClientAppID, serviceProperties : in TpServicePropertyList, serviceInstanceID : in TpServiceInstanceID) : IpServiceRef destroyServiceManager (serviceInstance : in TpServiceInstanceID) : void </pre>

8.3.2.1.1 Method createServiceManager()

This method returns a new service manager interface reference for the specified application. The service instance will be configured for the client application using the properties agreed in the service level agreement.

In case there is already a service manager available for the specified application and serviceInstanceID this reference is returned and no new service manager is created.

Returns <serviceManager> : Specifies the service manager interface reference for the specified application ID.

Parameters

application : in TpClientAppID

Specifies the application for which the service manager interface is requested.

serviceProperties : in TpServicePropertyList

Specifies the service properties and their values that are to be used to configure the service instance. These properties form a part of the service level agreement. An example of these properties is a list of methods that the client application is allowed to invoke on the service interfaces.

serviceInstanceID : in TpServiceInstanceID

Specifies the Service Instance ID that the new Service Manager is to be identified by.

Returns

IpServiceRef

Raises

TpCommonExceptions, P_INVALID_PROPERTY

8.3.2.1.2 Method destroyServiceManager()

This method destroys an existing service manager interface reference. This will result in the client application being unable to use the service manager any more.

*Parameters***serviceInstance** : in TpServiceInstanceID

Identifies the Service Instance to be destroyed.

*Raises***TpCommonExceptions**

8.3.3 Service Discovery Interface Classes

This API complements the Service Registration functionality described in another clause.

Before a service can be registered in the framework, the service supplier must know what "types" of services the Framework supports and what service "properties" are applicable to each service type. The "listServiceType()" method returns a list of all "service types" that are currently supported by the framework and the "describeServiceType()" method returns a description of each service type. The description of service type includes the "service-specific properties" that are applicable to each service type. Then the service supplier can retrieve a specific set of registered services that both belong to a given type and possess a specific set of "property values", by using the "discoverService()" method.

Additionally the service supplier can retrieve a list of all registered services, without regard to type or property values, by using the "listRegisteredServices()" method. However the scope of the list will depend upon the framework implementation; e.g. a service supplier may only be permitted to retrieve a list of services that the service supplier has previously registered.

8.3.3.1 Interface Class IpFwServiceDiscovery

Inherits from: IpInterface.

This interface shall be implemented by a Framework with as a minimum requirement the listServiceTypes(), describeServiceType() and discoverService() methods.

<<Interface>> IpFwServiceDiscovery
listServiceTypes () : TpServiceTypeNameList describeServiceType (name : in TpServiceTypeName) : TpServiceTypeDescription discoverService (serviceName : in TpServiceTypeName, desiredPropertyList : in TpServicePropertyList, max : in TpInt32) : TpServiceList listRegisteredServices () : TpServiceList

8.3.3.1.1 Method listServiceTypes()

This operation returns the names of all service types that are in the repository. The details of the service types can then be obtained using the describeServiceType() method.

Returns <listTypes> : The names of the requested service types.

Parameters

No Parameters were identified for this method.

Returns

TpServiceTypeNameList

Raises

TpCommonExceptions

8.3.3.1.2 Method describeServiceType()

This operation lets the caller obtain the details for a particular service type.

Returns <serviceTypeDescription> : The description of the specified service type. The description provides information about: the service properties associated with this service type: i.e. a list of service property {name, mode and type} tuples, the names of the super types of this service type, and whether the service type is currently available or unavailable.

Parameters

name : in TpServiceTypeName

The name of the service type to be described. If the "name" is malformed, then the P_ILLEGAL_SERVICE_TYPE exception is raised. If the "name" does not exist in the repository, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

Returns

TpServiceTypeDescription

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE

8.3.3.1.3 Method discoverService()

The discoverService operation is the means by which the service supplier can retrieve a specific set of registered services that both belong to a given type and possess a specific set of "property values". The service supplier passes in a list of desired service properties to describe the service it is looking for, in the form of attribute/value pairs for the service properties. The service supplier also specifies the maximum number of matched responses it is willing to accept. The framework must not return more matches than the specified maximum, but it is up to the discretion of the Framework implementation to choose to return less than the specified maximum. The discoverService() operation returns a serviceID/Property pair list for those services that match the desired service property list that the service supplier provided.

Returns <serviceList> : This parameter gives a list of matching services. Each service is characterised by its service ID and a list of service properties {name and value list} associated with the service.

Parameters

serviceName : in TpServiceTypeName

The name of the required service type. If the string representation of the "type" does not obey the rules for service type identifiers, then the P_ILLEGAL_SERVICE_TYPE exception is raised. If the "type" is correct syntactically but is not recognised as a service type within the Framework, then the P_UNKNOWN_SERVICE_TYPE exception is raised. The framework may return a service of a subtype of the "type" requested. A service sub-type can be described by the properties of its supertypes.

desiredPropertyList : in TpServicePropertyList

The "desiredPropertyList" parameter is a list of service properties {name and value list} that the required services should satisfy. These properties deal with the non-functional and non-computational aspects of the desired service. The property values in the desired property list must be logically interpreted as "minimum", "maximum", etc. by the framework (due to the absence of a Boolean constraint expression for the specification of the service criterion). It is suggested that, at the time of service registration, each property value be specified as an appropriate range of values, so that desired property values can specify an "enclosing" range of values to help in the selection of desired services.

max : in TpInt32

The "max" parameter states the maximum number of services that are to be returned in the "serviceList" result.

Returns

TpServiceList

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE, P_INVALID_PROPERTY

8.3.3.1.4 Method listRegisteredServices()

Returns a list of services so far registered in the framework.

Returns <serviceList> : The "serviceList" parameter returns a list of registered services. Each service is characterised by its service ID and a list of service properties {name and value list} associated with the service.

Parameters

No Parameters were identified for this method.

Returns

TpServiceList

Raises

TpCommonExceptions

8.3.4 Integrity Management Interface Classes

8.3.4.1 Interface Class IpFwFaultManager

Inherits from: IpInterface.

This interface is used by the service instance to inform the framework of events which affect the integrity of the API, and request fault management status information from the framework. The fault manager operations do not exchange callback interfaces as it is assumed that the service instance has supplied its Fault Management callback interface at the time it obtains the Framework's Fault Management interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.

If the IpFwFaultManager interface is implemented by a Framework, at least one of these methods shall be implemented. If the Framework is capable of invoking the IpSvcFaultManager.svcActivityTestReq() method, it shall implement svcActivityTestRes() and svcActivityTestErr() in this interface. If the Framework is capable of invoking IpSvcFaultManager.generateFaultStatisticsRecordReq(), it shall implement generateFaultStatisticsRecordRes() and generateFaultStatisticsRecordErr() in this interface. If the Framework is capable of invoking IpSvcFaultManager.generateFaultStatisticsRecordReq(), it shall implement generateFaultStatisticsRecordRes() and generateFaultStatisticsRecordErr() in this interface.

<<Interface>> IpFwFaultManager
activityTestReq (activityTestID : in TpActivityTestID, testSubject : in TpSubjectType) : void svcActivityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void appUnavailableInd () : void svcActivityTestErr (activityTestID : in TpActivityTestID) : void svcAvailStatusInd (reason : in TpSvcAvailStatusReason) : void generateFaultStatisticsRecordReq (faultStatsReqID : in TpFaultReqID, timePeriod : in TpTimeInterval, recordSubject : in TpSubjectType) : void generateFaultStatisticsRecordRes (faultStatsReqID : in TpFaultReqID, faultStatistics : in TpFaultStatsRecord) : void generateFaultStatisticsRecordErr (faultStatsReqID : in TpFaultReqID, faultStatisticsError : in TpFaultStatisticsError) : void

8.3.4.1.1 Method activityTestReq()

The service instance invokes this method to test that the framework or the client application is operational. On receipt of this request, the framework must carry out a test on itself or on the application, to check that it is operating correctly. The framework reports the test result by invoking the activityTestRes method on the IpSvcFaultManager interface.

Parameters

activityTestID : in TpActivityTestID

The identifier provided by the service instance to correlate the response (when it arrives) with this request.

testSubject : in TpSubjectType

Identifies the subject for testing (framework or client application).

Raises

TpCommonExceptions

8.3.4.1.2 Method svcActivityTestRes()

The service instance uses this method to return the result of a framework-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the framework to correlate this response (when it arrives) with the original request.

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

8.3.4.1.3 Method appUnavailableInd()

This method is used by the service instance to inform the framework that the client application is not responding. On receipt of this indication, the framework must act to inform the client application.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.3.4.1.4 Method svcActivityTestErr()

The service instance uses this method to indicate that an error occurred during a framework-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the framework to correlate this response (when it arrives) with the original request.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

8.3.4.1.5 Method svcAvailStatusInd()

This method is used by the service instance to inform the framework that it is about to become unavailable for use according to the provided reason and as well to inform the Framework when the Service instance becomes available again. The framework should inform the client applications that are currently using this service instance that it is unavailable and as well when it becomes available again for use (via the svcAvailStatusInd method on the IpAppFaultManager interface).

Parameters

reason : in TpSvcAvailStatusReason

Identifies the reason for the service instance's unavailability and also the reason SERVICE_AVAILABLE to be used to inform the Framework when the Service instance becomes available again.

Raises

TpCommonExceptions

8.3.4.1.6 Method generateFaultStatisticsRecordReq()

This method is used by the service instance to solicit fault statistics from the framework. On receipt of this request, the framework must produce a fault statistics record, for the framework or for the application during the specified time interval, which is returned to the service instance using the generateFaultStatisticsRecordRes operation on the IpSvcFaultManager interface.

Parameters

faultStatsReqID : in TpFaultReqID

The identifier provided by the service instance to correlate the response (when it arrives) with this request.

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. Supplying both a start time and stop time as empty strings leaves the time period to the discretion of the framework.

recordSubject : in TpSubjectType

Specifies the subject to be included in the general fault statistics record (framework or application).

Raises

TpCommonExceptions

8.3.4.1.7 Method generateFaultStatisticsRecordRes()

This method is used by the service to provide fault statistics to the framework in response to a generateFaultStatisticsRecordReq method invocation on the IpSvcFaultManager interface.

Parameters

faultStatsReqID : in TpFaultReqID

Used by the framework to correlate this response (when it arrives) with the original request.

faultStatistics : in **TpFaultStatsRecord**

The fault statistics record.

Raises

TpCommonExceptions

8.3.4.1.8 Method generateFaultStatisticsRecordErr()

This method is used by the service to indicate an error fulfilling the request to provide fault statistics, in response to a generateFaultStatisticsRecordReq method invocation on the IpSvcFaultManager interface.

Parameters

faultStatsReqID : in **TpFaultReqID**

Used by the framework to correlate this error (when it arrives) with the original request.

faultStatisticsError : in **TpFaultStatisticsError**

The fault statistics error.

Raises

TpCommonExceptions

8.3.4.2 Interface Class IpSvcFaultManager

Inherits from: IpInterface.

This interface is used to inform the service instance of events that affect the integrity of the Framework, Service or Client Application. The Framework will invoke methods on the Fault Management Service Interface that is specified when the service instance obtains the Fault Management Framework interface: i.e. by use of the obtainInterfaceWithCallback operation on the IpAccess interface.

If the IpSvcFaultManager interface is implemented by a Service, at least one of these methods shall be implemented. If the Service is capable of invoking the IpFwFaultManager.activityTestReq() method, it shall implement activityTestRes() and activityTestErr() in this interface. If the Service is capable of invoking IpFwFaultManager.generateFaultStatisticsRecordReq(), it shall implement generateFaultStatisticsRecordRes() and generateFaultStatisticsRecordErr() in this interface.

<<Interface>> IpSvcFaultManager
activityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void svcActivityTestReq (activityTestID : in TpActivityTestID) : void <<deprecated>> fwFaultReportInd (fault : in TpInterfaceFault) : void <<deprecated>> fwFaultRecoveryInd (fault : in TpInterfaceFault) : void <<deprecated>> fwUnavailableInd (reason : in TpFwUnavailReason) : void svcUnavailableInd () : void activityTestErr (activityTestID : in TpActivityTestID) : void appAvailStatusInd (reason : in TpAppAvailStatusReason) : void generateFaultStatisticsRecordRes (faultStatsReqID : in TpFaultReqID, faultStatistics : in TpFaultStatsRecord, recordSubject : in TpSubjectType) : void generateFaultStatisticsRecordErr (faultStatsReqID : in TpFaultReqID, faultStatisticsError : in TpFaultStatisticsError, recordSubject : in TpSubjectType) : void generateFaultStatisticsRecordReq (faultStatsReqID : in TpFaultReqID, timePeriod : in TpTimeInterval) : void fwAvailStatusInd (reason : in TpFwAvailStatusReason) : void

8.3.4.2.1 Method activityTestRes()

The framework uses this method to return the result of a service-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the service to correlate this response (when it arrives) with the original request.

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

8.3.4.2.2 Method svcActivityTestReq()

The framework invokes this method to test that the service instance is operational. On receipt of this request, the service instance must carry out a test on itself, to check that it is operating correctly. The service instance reports the test result by invoking the svcActivityTestRes method on the IpFwFaultManager interface.

*Parameters***activityTestID** : in **TpActivityTestID**

The identifier provided by the framework to correlate the response (when it arrives) with this request.

*Raises***TpCommonExceptions****8.3.4.2.3** Method <<deprecated>> fwFaultReportInd()

This method is deprecated and will be removed in a later release. It is strongly recommended not to implement this method. The new method fwAvailStatusInd shall be used instead, using the new type of reason parameter to inform the Service the reason why the Framework is unavailable.

The framework invokes this method to notify the service instance of a failure within the framework. The service instance must not continue to use the framework until it has recovered (as indicated by a fwFaultRecoveryInd).

*Parameters***fault** : in **TpInterfaceFault**

Specifies the fault that has been detected by the framework.

*Raises***TpCommonExceptions****8.3.4.2.4** Method <<deprecated>> fwFaultRecoveryInd()

This method is deprecated and will be removed in a later release. It is strongly recommended not to implement this method. The new method fwAvailStatusInd shall be used instead, using the new type of reason parameter to inform the Service when the Framework becomes available again.

The framework invokes this method to notify the service instance that a previously reported fault has been rectified. The service instance may then resume using the framework.

*Parameters***fault** : in **TpInterfaceFault**

Specifies the fault from which the framework has recovered.

*Raises***TpCommonExceptions**

8.3.4.2.5 Method <<deprecated>> fwUnavailableInd()

This method is deprecated and will be removed in a later release. It is strongly recommended not to implement this method. The new method fwAvailStatusInd shall be used instead, using the new type of reason parameter to inform the Application the reason why the Framework is unavailable and also when the Framework becomes available again.

The framework invokes this method to inform the service instance that it is no longer available.

Parameters

reason : in TpFwUnavailReason

Identifies the reason why the framework is no longer available.

Raises

TpCommonExceptions

8.3.4.2.6 Method svcUnavailableInd()

The framework invokes this method to inform the service instance that the client application has reported that it can no longer use the service instance.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.3.4.2.7 Method activityTestErr()

The framework uses this method to indicate that an error occurred during a service-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the service instance to correlate this response (when it arrives) with the original request.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

8.3.4.2.8 Method appAvailStatusInd()

The framework invokes this method to inform the service instance that the client application is no longer available using different reasons for the unavailability. This may be a result of the application reporting a failure. Alternatively, the framework may have detected that the application has failed: e.g. non-response from an activity test, failure to return

heartbeats, using the reason APP_UNAVAILABLE_NO_RESPONSE. When the application becomes available again the reason APP_AVAILABLE shall be used to inform the Service about that.

Parameters

reason : in TpAppAvailStatusReason

Identifies the reason why the application is no longer available. APP_AVAILABLE is used to inform the Service that the Application is available again.

Raises

TpCommonExceptions

8.3.4.2.9 Method generateFaultStatisticsRecordRes()

This method is used by the framework to provide fault statistics to a service instance in response to a generateFaultStatisticsRecordReq method invocation on the IpFwFaultManager interface.

Parameters

faultStatsReqID : in TpFaultReqID

Used by the service instance to correlate this response (when it arrives) with the original request.

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

recordSubject : in TpSubjectType

Specifies the entity (framework or application) whose fault statistics record has been provided.

Raises

TpCommonExceptions

8.3.4.2.10 Method generateFaultStatisticsRecordErr()

This method is used by the framework to indicate an error fulfilling the request to provide fault statistics, in response to a generateFaultStatisticsRecordReq method invocation on the IpFwFaultManager interface.

Parameters

faultStatsReqID : in TpFaultReqID

Used by the service instance to correlate this error (when it arrives) with the original request.

faultStatisticsError : in TpFaultStatisticsError

The fault statistics error.

recordSubject : in TpSubjectType

Specifies the entity (framework or application) whose fault statistics record was requested.

Raises

TpCommonExceptions

8.3.4.2.11 Method generateFaultStatisticsRecordReq()

This method is used by the framework to solicit fault statistics from the service instance, for example when the framework was asked for these statistics by the client application using the generateFaultStatisticsRecordReq operation on the IpFaultManager interface. On receipt of this request the service instance must produce a fault statistics record during the specified time interval, which is returned to the framework using the generateFaultStatisticsRecordRes operation on the IpFwFaultManager interface.

Parameters

faultStatsReqID : in TpFaultReqID

The identifier provided by the framework to correlate the response (when it arrives) with this request.

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. Supplying both a start time and stop time as empty strings leaves the time period to the discretion of the service.

Raises

TpCommonExceptions

8.3.4.2.12 Method fwAvailStatusInd()

The framework invokes this method to inform the service instance about the Framework availability status, i.e. that it can no longer use the Framework according to the reason parameter or that the Framework has become available again. The service instance may wait for the problem to be solved and just stop the usage of the Framework until the fwAvailStatusInd() is called again with the reason FRAMEWORK_AVAILABLE.

Parameters

reason : in TpFwAvailStatusReason

Identifies the reason why the framework is no longer available or that it has become available again.

8.3.4.3 Interface Class IpFwHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the framework by a service instance. If the IpFwHeartBeatMgmt interface is implemented by a Framework, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.

<<Interface>> IpFwHeartBeatMgmt
enableHeartBeat (interval : in TpInt32, svcInterface : in IpSvcHeartBeatRef) : void disableHeartBeat () : void changeInterval (interval : in TpInt32) : void

8.3.4.3.1 Method enableHeartBeat()

With this method, the service instance instructs the framework to begin sending its heartbeat to the specified interface at the specified interval.

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

svcInterface : in IpSvcHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

Raises

TpCommonExceptions, P_INVALID_INTERFACE_TYPE

8.3.4.3.2 Method disableHeartBeat()

Instructs the framework to cease the sending of its heartbeat.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.3.4.3.3 Method changeInterval()

Allows the administrative change of the heartbeat interval.

*Parameters***interval : in TpInt32**

The time interval in milliseconds between the heartbeats.

*Raises***TpCommonExceptions****8.3.4.4 Interface Class IpFwHeartBeat**

Inherits from: IpInterface.

The service side framework heartbeat interface is used by the service instance to send the framework its heartbeat. If a Framework is capable of invoking IpSvcHeartBeatMgmt.enableHeartBeat(), it shall implement IpFwHeartBeat and the pulse() method.

<<Interface>> IpFwHeartBeat
pulse () : void

8.3.4.4.1 Method pulse()

The service instance uses this method to send its heartbeat to the framework. The framework will be expecting a pulse at the end of every interval specified in the parameter to the IpSvcHeartBeatMgmt.enableSvcHeartbeat() method. If the pulse() is not received within the specified interval, then the service instance can be deemed to have failed the heartbeat.

Parameters

No Parameters were identified for this method.

*Raises***TpCommonExceptions****8.3.4.5 Interface Class IpSvcHeartBeatMgmt**

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the service instance by the framework. If the IpSvcHeartBeatMgmt interface is implemented by a Service, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.

<<Interface>> IpSvcHeartBeatMgmt
enableSvcHeartBeat (interval : in TpInt32, fwInterface : in IpFwHeartBeatRef) : void disableSvcHeartBeat () : void changeInterval (interval : in TpInt32) : void

8.3.4.5.1 Method enableSvcHeartBeat()

With this method, the framework instructs the service instance to begin sending its heartbeat to the specified interface at the specified interval.

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

fwInterface : in IpFwHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

Raises

TpCommonExceptions, P_INVALID_INTERFACE_TYPE

8.3.4.5.2 Method disableSvcHeartBeat()

Instructs the service instance to cease the sending of its heartbeat.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.3.4.5.3 Method changeInterval()

Allows the administrative change of the heartbeat interval.

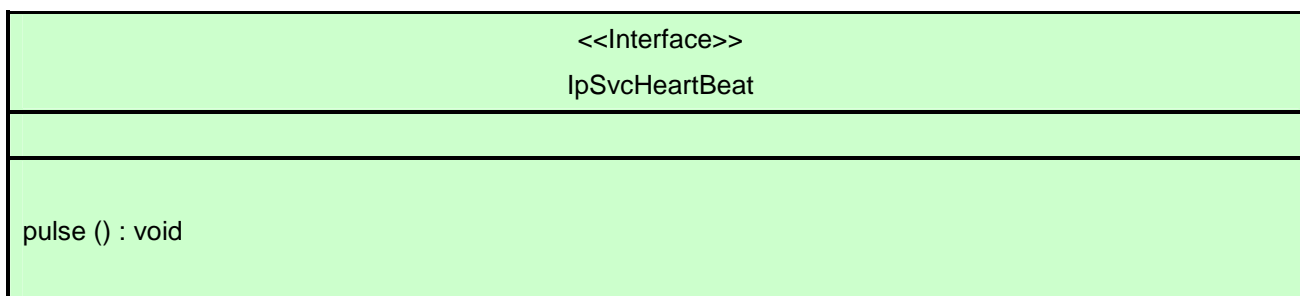
*Parameters***interval : in TpInt32**

The time interval in milliseconds between the heartbeats.

*Raises***TpCommonExceptions****8.3.4.6 Interface Class IpSvcHeartBeat**

Inherits from: IpInterface.

The service heartbeat interface is used by the framework to send the service instance its heartbeat. If a Service is capable of invoking IpFwHeartBeatMgmt.enableHeartBeat(), it shall implement IpSvcHeartBeat and the pulse() method.

**8.3.4.6.1 Method pulse()**

The framework uses this method to send its heartbeat to the service instance. The service will be expecting a pulse at the end of every interval specified in the parameter to the IpFwHeartBeatMgmt.enableHeartbeat() method. If the pulse() is not received within the specified interval, then the framework can be deemed to have failed the heartbeat.

Parameters

No Parameters were identified for this method.

*Raises***TpCommonExceptions****8.3.4.7 Interface Class IpFwLoadManager**

Inherits from: IpInterface.

The framework API should allow the load to be distributed across multiple machines and across multiple component processes, according to a load management policy. The separation of the load management mechanism and load management policy ensures the flexibility of the load management services. The load management policy identifies what load management rules the framework should follow for the specific service. It might specify what action the framework should take as the congestion level changes. For example, some real-time critical applications will want to make sure continuous service is maintained, below a given congestion level, at all costs, whereas other services will be satisfied with disconnecting and trying again later if the congestion level rises. Clearly, the load management policy is related to the QoS level to which the application is subscribed. The framework load management function is represented by the IpFwLoadManager interface. To handle responses and reports, the service developer must implement the

IpSvcLoadManager interface to provide the callback mechanism.

If the IpFwLoadManager interface is implemented by a Framework, at least one of the methods shall be implemented as a minimum requirement. If load level notifications are supported, the createLoadLevelNotification() and destroyLoadLevelNotification() methods shall be implemented. If suspendNotification() is implemented, then resumeNotification() shall be implemented also. If a Framework is capable of invoking the IpSvcLoadManager.querySvcLoadStatsReq() method, then it shall implement querySvcLoadStatsRes() and querySvcLoadStatsErr() methods in this interface.

<<Interface>> IpFwLoadManager
<pre> reportLoad (loadLevel : in TpLoadLevel) : void createLoadLevelNotification (notificationSubject : in TpSubjectType) : void destroyLoadLevelNotification (notificationSubject : in TpSubjectType) : void suspendNotification (notificationSubject : in TpSubjectType) : void resumeNotification (notificationSubject : in TpSubjectType) : void queryLoadStatsReq (loadStatsReqID : in TpLoadTestID, querySubject : in TpSubjectType, timeInterval : in TpTimeInterval) : void querySvcLoadStatsRes (loadStatsReqID : in TpLoadTestID, loadStatistics : in TpLoadStatisticList) : void querySvcLoadStatsErr (loadStatsReqID : in TpLoadTestID, loadStatisticError : in TpLoadStatisticError) : void </pre>

8.3.4.7.1 Method reportLoad()

The service instance uses this method to report its current load level (0, 1, or 2) to the framework: e.g. when the load level on the service instance has changed.

At level 0 load, the service instance is performing within its load specifications (i.e. it is not congested or overloaded). At level 1 load, the service instance is overloaded. At level 2 load, the service instance is severely overloaded. In addition this method shall be called by the service instance in order to report current load status, when load notifications are first requested, or resumed after suspension.

Parameters

loadLevel : in TpLoadLevel

Specifies the service instance's load level.

Raises

TpCommonExceptions

8.3.4.7.2 Method createLoadLevelNotification()

The service instance uses this method to register to receive notifications of load level changes associated with the framework or with the application that uses the service instance. Upon receipt of this method the framework shall inform the service instance of the current framework or application load using the loadLevelNotification method on the corresponding IpSvcLoadManager.

Parameters

notificationSubject : in TpSubjectType

Specifies the entity (framework or application) for which load level changes should be reported.

Raises

TpCommonExceptions

8.3.4.7.3 Method destroyLoadLevelNotification()

The service instance uses this method to unregister for notifications of load level changes associated with the framework or with the application that uses the service instance.

Parameters

notificationSubject : in TpSubjectType

Specifies the entity (framework or application) for which load level changes should no longer be reported.

Raises

TpCommonExceptions

8.3.4.7.4 Method suspendNotification()

The service instance uses this method to request the framework to suspend sending it notifications associated with the framework or with the application that uses the service instance; e.g. while the service instance handles a temporary overload condition.

Parameters

notificationSubject : in TpSubjectType

Specifies the entity (framework or application) for which the sending of notifications by the framework should be suspended.

Raises

TpCommonExceptions

8.3.4.7.5 Method resumeNotification()

The service instance uses this method to request the framework to resume sending its notifications associated with the framework or with the application that uses the service instance; e.g. after a period of suspension during which the service instance handled a temporary overload condition. Upon receipt of this method the framework shall inform the service instance of the current framework or application load using the loadLevelNotification method on the corresponding IpSvcLoadManager.

Parameters

notificationSubject : in TpSubjectType

Specifies the entity (framework or application) for which the sending of notifications of load level changes by the framework should be resumed.

Raises

TpCommonExceptions

8.3.4.7.6 Method queryLoadStatsReq()

The service instance uses this method to request the framework to provide load statistics records for the framework or for the application that uses the service instance.

Parameters

loadStatsReqID : in TpLoadTestID

The identifier provided by the service instance to correlate the response (when it arrives) with this request.

querySubject : in TpSubjectType

Specifies the entity (framework or application) for which load statistics records should be reported.

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistics records should be reported.

Raises

TpCommonExceptions

8.3.4.7.7 Method querySvcLoadStatsRes()

The service instance uses this method to send load statistic records back to the framework that requested the information; i.e. in response to an invocation of the querySvcLoadStatsReq method on the IpSvcLoadManager interface.

Parameters

loadStatsReqID : in TpLoadTestID

Used by the framework to correlate this response (when it arrives) with the original request.

loadStatistics : in TpLoadStatisticList

Specifies the service-supplied load statistics.

*Raises***TpCommonExceptions****8.3.4.7.8 Method querySvcLoadStatsErr()**

The service instance uses this method to return an error response to the framework that requested the service instance's load statistics information, when the service instance is unsuccessful in obtaining any load statistic records; i.e. in response to an invocation of the querySvcLoadStatsReq method on the IpSvcLoadManager interface.

Parameters

loadStatsReqID : in TpLoadTestID

Used by the framework to correlate this error (when it arrives) with the original request.

loadStatisticError : in TpLoadStatisticError

Specifies the error code associated with the failed attempt to retrieve the service instance's load statistics.

*Raises***TpCommonExceptions****8.3.4.8 Interface Class IpSvcLoadManager**

Inherits from: IpInterface.

The service developer supplies the load manager service interface to handle requests, reports and other responses from the framework load manager function. The service instance supplies the identity of its callback interface at the time it obtains the framework's load manager interface, by use of the obtainInterfaceWithCallback() method on the IpAccess interface.

If the IpSvcLoadManager interface is implemented by a Service, at least one of the methods shall be implemented as a minimum requirement. If load level notifications are supported, then loadLevelNotification() shall be implemented. If a Service is capable of invoking the IpFwLoadManager.queryLoadStatsReq() method, then it shall implement queryLoadStatsRes() and queryLoadStatsErr() methods in this interface.

<<Interface>> IpSvcLoadManager
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void suspendNotification () : void resumeNotification () : void createLoadLevelNotification () : void destroyLoadLevelNotification () : void querySvcLoadStatsReq (loadStatsReqID : in TpLoadTestID, timeInterval : in TpTimeInterval) : void queryLoadStatsRes (loadStatsReqID : in TpLoadTestID, loadStatistics : in TpLoadStatisticList) : void queryLoadStatsErr (loadStatsReqID : in TpLoadTestID, loadStatisticsError : in TpLoadStatisticError) : void

8.3.4.8.1 Method loadLevelNotification()

Upon detecting load condition change, (e.g. load level changing from 0 to 1, 0 to 2, 1 to 0, for the application or framework which has been registered for load level notifications) this method is invoked on the SCF. In addition this method shall be invoked on the SCF in order to provide a notification of current load status, when load notifications are first requested, or resumed after suspension.

Parameters

loadStatistics : in **TpLoadStatisticList**

Specifies the framework-supplied load statistics, which include the load level change(s).

Raises

TpCommonExceptions

8.3.4.8.2 Method suspendNotification()

The framework uses this method to request the service instance to suspend sending it any notifications: e.g. while the framework handles a temporary overload condition.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.3.4.8.3 Method resumeNotification()

The framework uses this method to request the service instance to resume sending it notifications: e.g. after a period of suspension during which the framework handled a temporary overload condition. Upon receipt of this method the service instance shall inform the framework of the current load using the reportLoad method on the corresponding IpFwLoadManager.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.3.4.8.4 Method createLoadLevelNotification()

The framework uses this method to register to receive notifications of load level changes associated with the service instance. Upon receipt of this method the service instance shall inform the framework of the current load using the reportLoad method on the corresponding IpFwLoadManager.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.3.4.8.5 Method destroyLoadLevelNotification()

The framework uses this method to unregister for notifications of load level changes associated with the service instance.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.3.4.8.6 Method querySvcLoadStatsReq()

The framework uses this method to request the service instance to provide its load statistic records.

Parameters

loadStatsReqID : in TpLoadTestID

The identifier provided by the framework to correlate the response (when it arrives) with this request.

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistic records should be reported.

Raises

TpCommonExceptions

8.3.4.8.7 Method queryLoadStatsRes()

The framework uses this method to send load statistic records back to the service instance that requested the information; i.e. in response to an invocation of the queryLoadStatsReq method on the IpFwLoadManager interface.

*Parameters***loadStatsReqID : in TpLoadTestID**

Used by the service instance to correlate this response (when it arrives) with the original request.

loadStatistics : in TpLoadStatisticList

Specifies the framework-supplied load statistics.

*Raises***TpCommonExceptions****8.3.4.8.8 Method queryLoadStatsErr()**

The framework uses this method to return an error response to the service that requested the framework's load statistics information, when the framework is unsuccessful in obtaining any load statistic records; i.e. in response to an invocation of the queryLoadStatsReq method on the IpFwLoadManager interface.

*Parameters***loadStatsReqID : in TpLoadTestID**

Used by the service instance to correlate this error (when it arrives) with the original request.

loadStatisticsError : in TpLoadStatisticError

Specifies the error code associated with the failed attempt to retrieve the framework's load statistics.

*Raises***TpCommonExceptions****8.3.4.9 Interface Class IpFwOAM**

Inherits from: IpInterface.

The OAM interface is used to query the system date and time. The service and the framework can synchronise the date and time to a certain extent. Accurate time synchronisation is outside the scope of this API. This interface and the systemDateTimeQuery() method are optional.

<<Interface>> IpFwOAM
systemDateTimeQuery (clientDateAndTime : in TpDateAndTime) : TpDateAndTime

8.3.4.9.1 Method systemDateTimeQuery()

This method is used to query the system date and time. The client (service) passes in its own date and time to the framework. The framework responds with the system date and time.

Returns <systemDateAndTime> : This is the system date and time of the framework.

Parameters

clientDateAndTime : in TpDateAndTime

This is the date and time of the client (service). The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Returns

TpDateAndTime

Raises

TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT

8.3.4.10 Interface Class IpSvcOAM

Inherits from: IpInterface.

This interface and the systemDateTimeQuery() method are optional.

<<Interface>> IpSvcOAM
systemDateTimeQuery (systemDateAndTime : in TpDateAndTime) : TpDateAndTime

8.3.4.10.1 Method systemDateTimeQuery()

This method is used by the framework to send the system date and time to the service. The service responds with its own date and time.

Returns <clientDateAndTime> : This is the date and time of the client (service).

Parameters

systemDateAndTime : in TpDateAndTime

This is the system date and time of the framework. The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Returns

TpDateAndTime

Raises

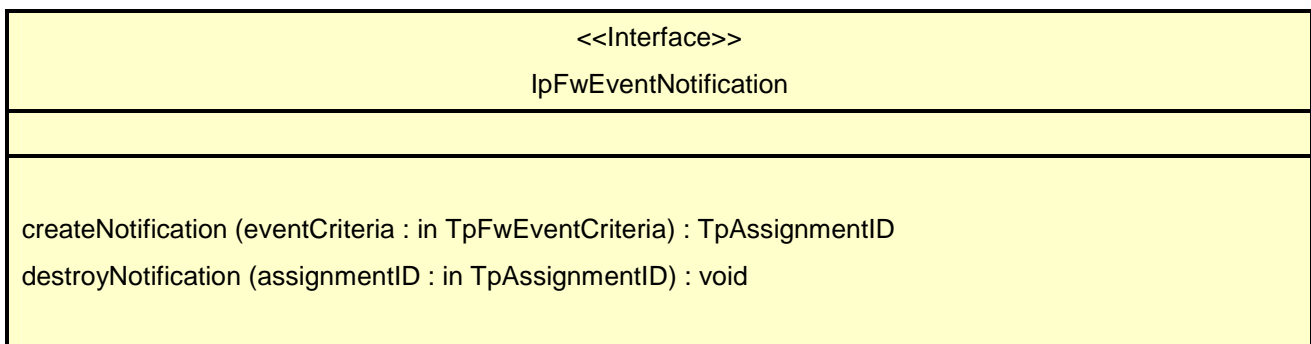
TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT

8.3.5 Event Notification Interface Classes

8.3.5.1 Interface Class IpFwEventNotification

Inherits from: IpInterface.

The event notification mechanism is used to notify the service of generic events that have occurred. If Event Notifications are supported by a Framework, this interface and the createNotification() and destroyNotification() methods shall be supported.



8.3.5.1.1 Method createNotification()

This method is used to install generic notifications so that events can be sent to the service.

Returns <assignmentID> : Specifies the ID assigned by the framework for this newly installed event notification.

Parameters

eventCriteria : in TpFwEventCriteria

Specifies the event specific criteria used by the service to define the event required.

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_INVALID_EVENT_TYPE, P_INVALID_CRITERIA

8.3.5.1.2 Method destroyNotification()

This method is used by the service to delete generic notifications from the framework.

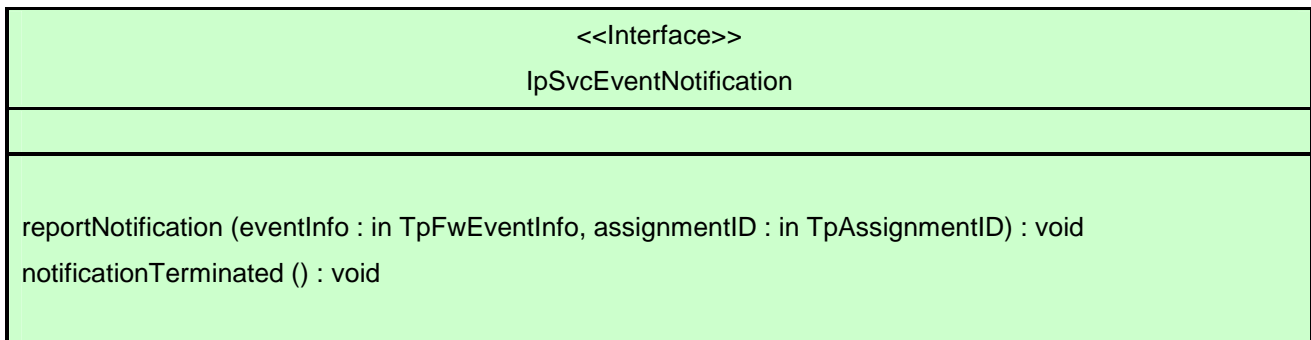
*Parameters***assignmentID : in TpAssignmentID**

Specifies the assignment ID given by the framework when the previous createNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENT_ID.

*Raises***TpCommonExceptions, P_INVALID_ASSIGNMENT_ID****8.3.5.2 Interface Class IpSvcEventNotification**

Inherits from: IpInterface.

This interface is used by the framework to inform the service of a generic event. The Event Notification Framework will invoke methods on the Event Notification Service Interface that is specified when the Event Notification interface is obtained. If Event Notifications are supported by a Service, this interface and the reportNotification() and notificationTerminated() methods shall be supported.

**8.3.5.2.1 Method reportNotification()**

This method notifies the service of the arrival of a generic event.

*Parameters***eventInfo : in TpFwEventInfo**

Specifies specific data associated with this event.

assignmentID : in TpAssignmentID

Specifies the assignment id which was returned by the framework during the createNotification() method. The service can use the assignment id to associate events with event specific criteria and to act accordingly.

*Raises***TpCommonExceptions, P_INVALID_ASSIGNMENT_ID**

8.3.5.2.2 Method notificationTerminated()

This method indicates to the service that all generic event notifications have been terminated (for example, due to faults detected).

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

8.4 State Transition Diagrams

This clause contains the State Transition Diagrams for the objects that implement the Framework interfaces on the gateway side. The State Transition Diagrams show the behaviour of these objects. For each state the methods that can be invoked by the client are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the client also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

8.4.1 Service Registration State Transition Diagrams

8.4.1.1 State Transition Diagrams for IpFwServiceRegistration

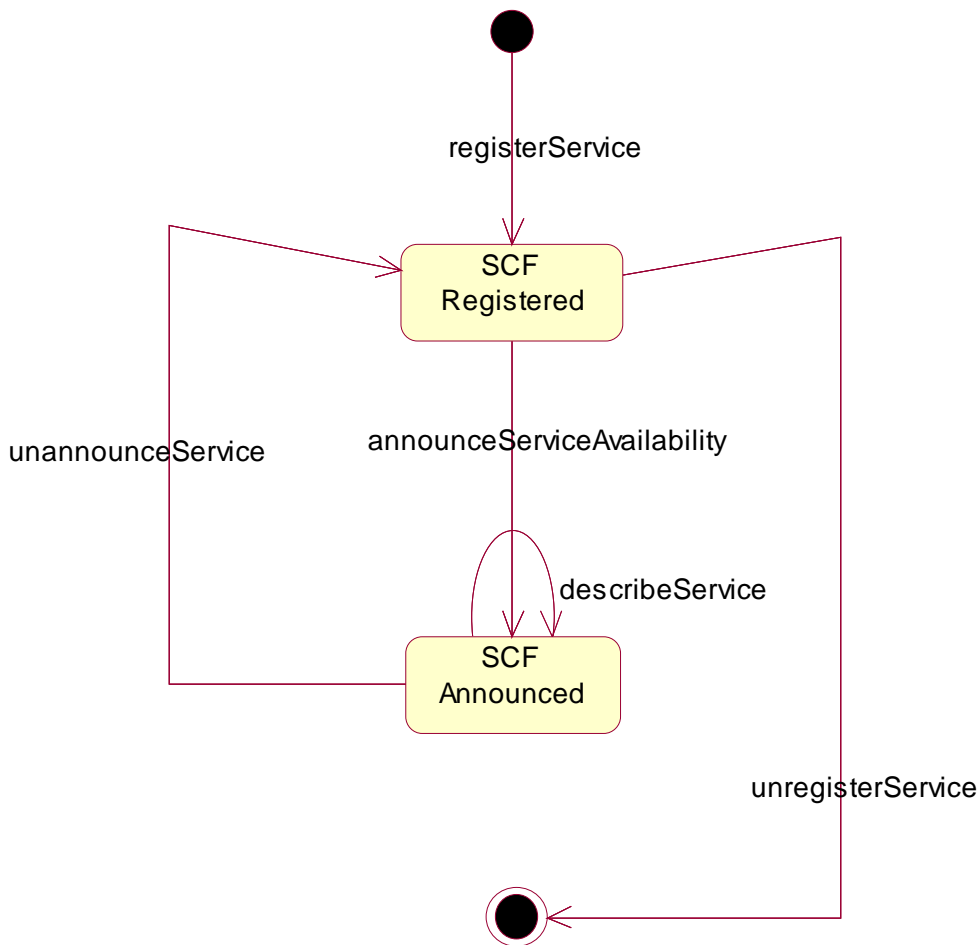


Figure : State Transition Diagram for IpFwServiceRegistration

8.4.1.1.1 SCF Registered State

This is the state entered when a Service Capability Server (SCS) registers its SCF in the Framework, by informing it of the existence of an SCF characterised by a service type and a set of service properties. As a result the Framework associates a service ID to this SCF, that will be used to identify it by both sides.

An SCF may be unregistered, the service ID then being no longer associated with the SCF.

8.4.1.1.2 SCF Announced State

This is the state entered when the existence of the SCF has been announced, thus making it available for discovery by applications. The SCF can be unannounced at any time, taking it back into the SCF Registered state where it is no longer available for discovery.

8.4.2 Service Instance Lifecycle Manager State Transition Diagrams

There are no State Transition Diagrams defined for Service Instance Lifecycle Manager

8.4.3 Service Discovery State Transition Diagrams

There are no State Transition Diagrams defined for Service Discovery

8.4.4 Integrity Management State Transition Diagrams

8.4.4.1 State Transition Diagrams for IpFwLoadManager

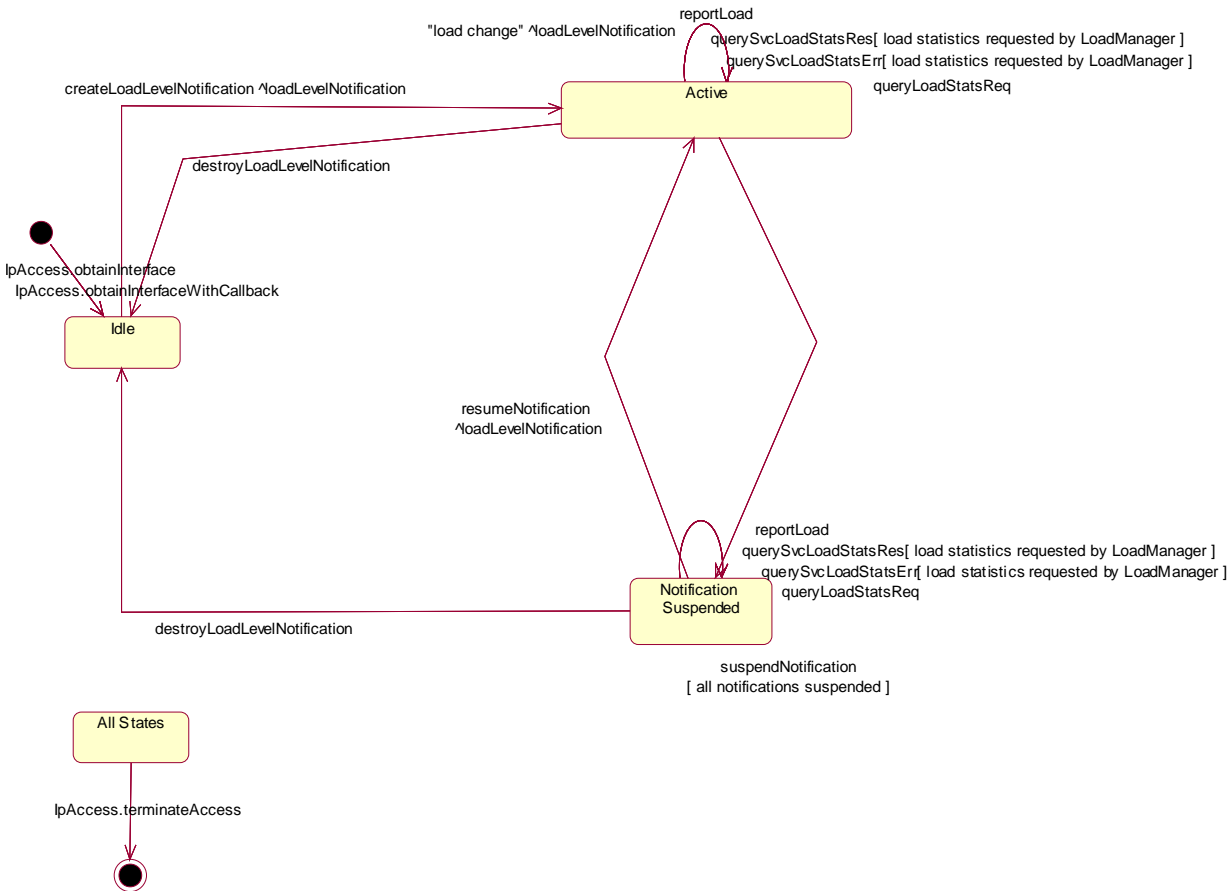


Figure : State Transition Diagram for IpFwLoadManager

8.4.4.1.1 Idle State

In this state the service has obtained an interface reference of the LoadManager from the IpAccess interface.

8.4.4.1.2 Notification Suspended State

Due to e.g. a temporary load condition, the service has requested the LoadManager to suspend sending the load level notification information.

8.4.4.1.3 Active State

In this state the service has indicated its interest in notifications by performing a `createLoadLevelNotification()` invocation on the IpFwLoadManager. The load manager can now request the service to supply load statistics information (by invoking `querySvcLoadStatsReq()`). Furthermore the LoadManager can request the service to control its load (by invoking `loadLevelNotification()`, `resumeNotification()` or `suspendNotification()` on the service side of interface). In case the service detects a change in load level, it reports this to the LoadManager by calling the method `reportLoad()`.

8.4.4.2 State Transition Diagrams for IpFwFaultManager

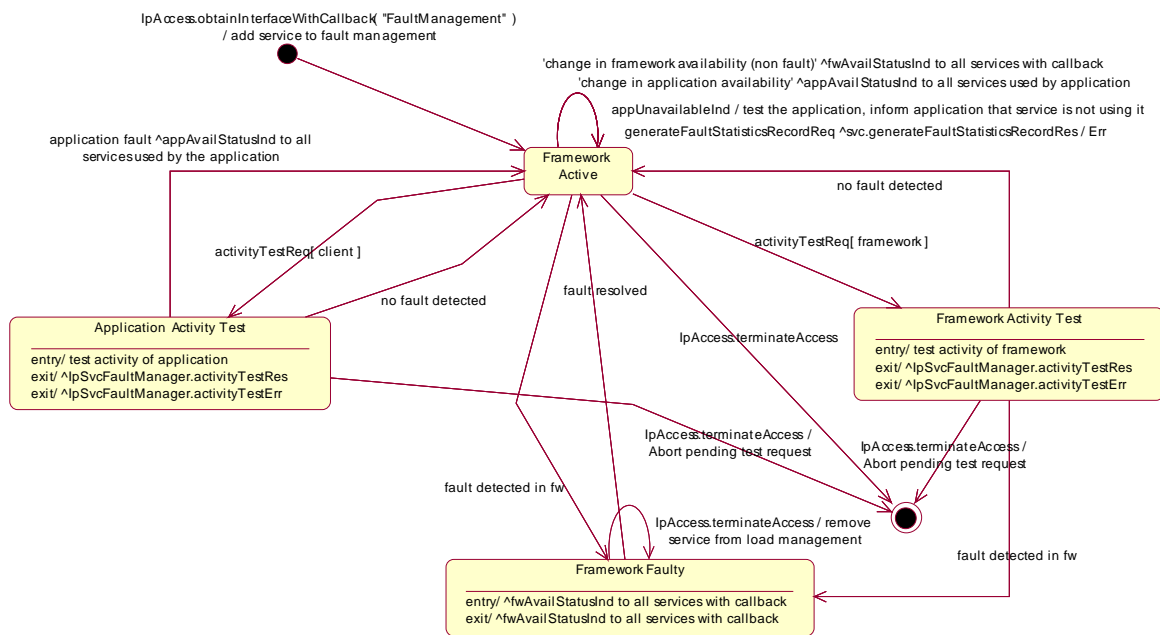


Figure : State Transition Diagram for IpFwFaultManager

8.4.4.2.1 Framework Active State

This is the normal state of the framework, which is fully functional and able to handle requests from both applications and service capability features.

8.4.4.2.2 Framework Activity Test State

In this state, the framework is performing a self-diagnostic test. If a problem is diagnosed, all services with fault management callbacks are notified through an fwAvailStatusInd message.

8.4.4.2.3 Application Activity Test State

In this state, the framework is performing a test on one client application. If the application is faulty, services that are used by the application and that have provided fault management callbacks are notified accordingly through an appAvailStatusInd message.

8.4.4.2.4 Framework Faulty State

In this state, the framework has detected an internal problem with itself such that application and service capability features cannot communicate with it anymore; attempts to invoke any methods that belong to any SCFs of the framework return an error. If the framework ever recovers, services with fault management callbacks will be notified via a fwAvailStatusInd message.

8.4.5 Event Notification State Transition Diagrams

There are no State Transition Diagrams defined for Event Notification

9 Service Properties

9.1 Service Super and Sub Types

Service Properties are used at service registration to indicate the capabilities of an SCF. They are normally used as an indication for limitations an SCF has. These limitations can come from the way an SCF is implemented or from limitations in the network. The service type of an SCF defines which properties the supplier shall provide at registration of the SCF.

An application uses Service Properties at service discovery to find services that have the required capabilities. The Framework validates the requested properties with the registered properties and provides the application with a list of SCFs that comply to the application's request.

The capabilities of an SCF can be extended by providing service properties in addition to the ones defined in this standard. For this extended SCF, a dedicated sub-type of a service is defined. A sub-type of an SCF shall be fully compatible with the standard SCF, that is, an application shall be able to use the sub type as if it was the standard type. This implies that the interface to the SCF remains unchanged. Also SCF sub types can be further extended. This way a hierarchy of service types can be built with the standard type being the root.

An example of a sub type is a Multy Party Call Control service that allows the application to request a certain quality-of-service level. An additional service property is added for this.

9.2 Service Property Types

At Service Registration the properties of a type shall be interpreted as the set of values that can be supported by the service. If a service type has a certain property (e.g. "CAN_DO_SOMETHING"), a service registers with a property value of {"true", "false"}. This means that the SCS is able to support Service instances where this property is used or allowed and instances where this property is not used or allowed. This clarifies why sets of values shall be used for the property values instead of primitive types.

At establishment of the Service Level Agreement the property can then be set to the value of the specific agreement. The context of the Service Level Agreement thus restricts the set of property values of the SCS and will thus lead to a sub-set of the service property values. When the correct SCF is instantiated during the discovery and selection procedure (see Note), the Service Properties shall thus be interpreted as the requested property values.

NOTE: This is achieved through the createServiceManager() operation in the Service Instance Lifecycle Manager interface.

All property values are represented by an array of strings. The following table shows all supported service property types.

Service Property type name	Description	Example value (array of strings)	Interpretation of example value
BOOLEAN_SET	set of Booleans	{"FALSE"}	The set of Booleans consisting of the Boolean "false".
INTEGER_SET	set of integers	{"1", "2", "5", "7"}	The set of integers consisting of the integers 1, 2, 5 and 7.
STRING_SET	set of strings	{"Sophia", "Rijen"}	The set of strings consisting of the string 'Sophia' and the string "Rijen"
INTEGER_INTERVAL	interval of integers	{"5", "100"}	The integers that are between or equal to 5 and 100.
STRING_INTERVAL	interval of strings	{"Rijen", "Sophia"}	The strings that are between or equal to the strings "Rijen" and "Sophia", in lexicographical order.
INTEGER_INTEGER_MAP	map from integers to integers	{"1", "10", "2", "20", "3", "30"}	The map that maps 1 to 10, 2 to 20 and 3 to 30.
XML_ADDRESS_RANGE_SET	set of values of TpAddressRange. Values of TpAddressRange are described using XML. An XML schema is provided below for this purpose.	<pre> { "<AddressRangeSet> <AddressRange> <Plan>P_ADDRESS_P LAN_E164</Plan> <AddrString>123*</Addr rString> </AddressRange> <AddressRange> <Plan>P_ADDRESS_P LAN_E164</Plan> <AddrString>456*</Addr rString> </AddressRange> </AddressRangeSet>" } </pre>	<p>In case P_REGEX_SUPPORT_FOR_ADDRESS_RANGE is TRUE: Any addresses containing 123 or containing 456 in the E.164 Address Plan.</p> <p>In case P_REGEX_SUPPORT_FOR_ADDRESS_RANGE is FALSE: Any addresses starting with 123 or starting with 456 in the E.164 Address Plan.</p>
FLOAT_SET	set of values of TpFloat.	{"0.1", '.2', '0.1e+3}	The set of floats containing floating point numbers 0.1, 0.2 and 100
FLOAT_INTERVAL	interval of TpFloat values	{"-1.1", '5.0'}	The floating point numbers that are between or equal to – 1.1 and 5.0

The bounds of the string interval, integer interval and float interval types may hold the reserved value "UNBOUNDED". If the left bound of the interval holds the value "UNBOUNDED", the lower bound of the interval is the smallest value supported by the type. If the right bound of the interval holds the value "UNBOUNDED", the upper bound of the interval is the largest value supported by the type.

When an SCF is registered by the Service Supplier, Service Properties of type BOOLEAN_SET shall not contain an empty set. When a service is discovered by an application, this application shall specify either {TRUE} or {FALSE} as value for service properties of type BOOLEAN_SET.

The value of XML_ADDRESS_RANGE_SET should comply with the following XML Schema:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="AddressRangeSet">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="AddressRange" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Plan" type="xs:string" default="P_ADDRESS_PLAN_ANY"/>
              <xs:element name="AddrString" type="xs:string"/>
              <xs:element name="Name" type="xs:string" minOccurs="0"/>
              <xs:element name="SubAddressString" type="xs:string" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

```

    </xs:complexType>
  </xs:element>
</xs:schema>

```

An example usage could be:

```

{ "<?xml version="1.0" encoding="UTF-8"?>
<AddressRangeSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml_address_range_set.xsd">
  <AddressRange>
    <Plan>P_ADDRESS_PLAN_E164</Plan>
    <AddrString>789* </AddrString>
  </AddressRange>
  <AddressRange>
    <Plan>P_ADDRESS_PLAN_ANY</Plan>
    <AddrString>123* </AddrString>
  </AddressRange>
  <AddressRange>
    <Plan>P_ADDRESS_PLAN_SIP</Plan>
    <AddrString>&lt;sip:*@parlay.org&gt;</AddrString>
    <Name/>
  </AddressRange>
</AddressRangeSet>"}

```

Note that the final address range corresponds to any sip address @parlay.org, i.e. < sip:*@parlay.org >.

9.3 General Service Properties

Each service instance has the following general properties:

- [Service Name](#)
- [Service Version](#)
- [Service ID](#)
- [Service Description](#)
- [Product Name](#)
- [Product Version](#)
- [Operation Set](#)
- Compatible Service
- Backward Compatibility Level
- Migration Required
- Data Migrated
- Migration Date and Time
- Support for Regular Expressions in Address Range

The following sections describe these general service properties in more detail. The values for the mode are defined in the type TpServiceTypePropertyMode.

9.3.1 Service Name

Property	Type	Mode	Description
P_SERVICE_NAME	STRING_SET	MANDATORY_READONLY	This property contains the name of the service, e.g. 'UserLocation', 'UserLocationCamel', 'UserLocationEmergency' or 'UserStatus'.

9.3.2 Service Version

Property	Type	Mode	Description
P_SERVICE_VERSION	STRING_SET	MANDATORY	This property contains the version of the APIs, to which the service is compliant. It is a set of strings as specified in the TpVersion type.

9.3.3 Service ID

Property	Type	Mode	Description
P_SERVICE_ID	STRING_INTERVAL	READONLY	This property uniquely identifies a specific service. Note that the Framework generates this property value when the Service Supplier registers the service. This property should not be confused with the serviceInstanceID generated by the Framework when a Client Application signs a Service Agreement to obtain the Service Manager

9.3.4 Service Description

Property	Type	Mode	Description
P_SERVICE_DESCRIPTION	STRING_SET	MANDATORY_READONLY	This property contains a textual description of the service. It should not be interpreted as a description of a Service Instance (as identified by a serviceInstanceID generated by the Framework when a Client Application signs a Service Agreement to obtain the Service Manager).

9.3.5 Product Name

Property	Type	Mode	Description
P_PRODUCT_NAME	STRING_SET	READONLY	This property contains the name of the product that provides the service, e.g. 'Find It', 'Locate.com'.

9.3.6 Product Version

Property	Type	Mode	Description
P_PRODUCT_VERSION	STRING_SET	READONLY	This property contains the version of the product that provides the service, e.g. '3.1.11'.

9.3.7 Void

9.3.8 Operation Set

Property	Type	Mode	Description
P_OPERATION_SET	STRING_SET	MANDATORY	Specifies set of the operations the SCS supports. The notation to be used is : {'Interface1.operation1','Interface1.operation 2', 'Interface2.operation1'}, e.g.: {'IpCall.createCall','IpCall.routeReq'}.

9.3.9 Compatible Service

Property	Type	Mode	Description
P_COMPATIBLE_WITH_SERVICE	STRING_SET	READONLY	Specifies the Set of Services, identified by their ServiceIDs, with which this new service is compatible. This property should at least be accompanied with the properties P_BACKWARD_COMPATIBILITY_LEVEL, P_MIGRATION_REQUIRED. Note that the new Service can be compatible with more than one Service that is currently registered to the Framework. Therefore this Property is a SET, as well as all related properties like Migration Required, Data Migrated, etc. For all these properties the order of the Services shall be identical.

9.3.10 Backward Compatibility Level

Property	Type	Mode	Description
P_BACKWARD_COMPATIBILITY_LEVEL	BOOLEAN_SET	READONLY	<p>Specifies if the new service is completely backwards compatible with each service identified in the P_COMPATIBLE_WITH_SERVICE property:</p> <p>Value = TRUE: Service is completely backwards compatible</p> <p>Value = FALSE: SCS is not completely backwards compatible.</p> <p>This property requires the presence of P_COMPATIBLE_WITH_SERVICE property.</p> <p>Note that the new Service can be compatible with more than one Service that is currently registered to the Framework. Therefore this Property is a SET, as well as all related properties.</p> <p>For each service identified in P_COMPATIBLE_WITH_SERVICE, one value of this property shall be present in the value set of this property at service registration.</p> <p>For all these properties the order of the Services shall be identical.</p>

9.3.11 Migration Required

Property	Type	Mode	Description
P_MIGRATION_REQUIRED	BOOLEAN_SET	READONLY	<p>Specifies if the new service is replacing the service identified in the P_COMPATIBLE_WITH_SERVICE property: Value = TRUE: new service is replacing the existing one – migration is required before the date/time indicated in P_MIGRATION_DATE_AND_TIME property. Value = FALSE: new service is not replacing the existing one – migration not required, the existing service is retained.</p> <p>This property requires the presence of P_COMPATIBLE_WITH_SERVICE property. If the value set of P_MIGRATION_REQUIRED contains TRUE, P_DATA_MIGRATED and P_MIGRATION_DATE_AND_TIME properties shall also be present.</p> <p>Note that the new Service can be compatible with more than one Service that is currently registered to the Framework. Therefore this Property is a SET, as well as all related properties.</p> <p>For each service identified in P_COMPATIBLE_WITH_SERVICE, one value of this property shall be present in the value set of this property at service registration. For all these properties the order of the Services shall be identical.</p>

9.3.12 Data Migrated

Property	Type	Mode	Description
P_DATA_MIGRATED	BOOLEAN_SET	READONLY	<p>Indicates if the data (e.g. notifications) from the existing service identified in the P_COMPATIBLE_WITH_SERVICE property is also available in this Service. Value = TRUE: all data is migrated Value = FALSE: no data is migrated</p> <p>This property requires the presence of P_COMPATIBLE_WITH_SERVICE and the P_MIGRATION_REQUIRED properties.</p> <p>Note that the new Service can be compatible with more than one Service that is currently registered to the Framework. Therefore this Property is a SET, as well as all related properties.</p> <p>For each service identified in P_COMPATIBLE_WITH_SERVICE, one value of this property shall be present in the value set of this property at service registration. For all these properties the order of the Services shall be identical.</p>

9.3.13 Migration Date And Time

Property	Type	Mode	Description
P_MIGRATION_DATE_AND_TIME	STRING_SET	READONLY	<p>This property contains the date and time, in the format described in TpDateAndTime, by which point applications shall have migrated from existing services to this new service. Migration to the new service requires the application to terminate the existing service agreement, and sign a new one. Failure to do this by the migration date and time indicated in this property may result in the service agreement being terminated by the Framework, since the service supplier may choose to unregister the service following this date and time. Only one value of TpDateAndTime is permitted to be present in this property at service registration.</p> <p>This property requires the presence of P_COMPATIBLE_WITH_SERVICE, P_MIGRATION_REQUIRED and P_DATA_MIGRATED properties.</p> <p>Note that the new Service can be compatible with more than one Service that is currently registered to the Framework. Therefore this Property is a SET, as well as all related properties.</p> <p>For each service identified in P_COMPATIBLE_WITH_SERVICE, one value of this property shall be present in the value set of this property at service registration.</p> <p>For all these properties the order of the Services shall be identical. For those services for which migration is not required (P_MIGRATION_REQUIRED set to FALSE), the corresponding value of this property shall be ignored.</p>

9.3.14 Support for Regular Expressions in Address Range

Property	Type	Mode	Description
P_REGEX_SUPPORT_FOR_ADDRESS_RANGE	BOOLEAN_SET	READONLY	Indicates if the AddrString and SubAddressString fields of TpAddressRange are expressed as regular expressions (TRUE) or not (FALSE)

10 Data Definitions

This clause provides the Framework specific data definitions necessary to support the OSA interface specification.

The general format of a data definition specification is the following:

- Data type, that shows the name of the data type;
- Description, that describes the data type;

- Tabular specification, that specifies the data types and values of the data type;
- Example, if relevant, shown to illustrate the data type.

All data types referenced but not defined in this clause are common data definitions which may be found in 3GPP TS 29.198-2.

10.1 Common Framework Data Definitions

10.1.1 TpClientAppID

This is an identifier for the client application. It is used to identify the client to the Framework. This data type is identical to TpString and is defined as a string of characters that uniquely identifies the application. The content of this string shall be unique for each OSA API implementation (or unique for a network operator's domain). This unique identifier shall be negotiated with the OSA operator and the application shall use it to identify itself.

10.1.2 TpClientAppIDList

This data type defines a Numbered Set of Data Elements of type TpClientAppID.

10.1.3 TpDomainID

Defines the Tagged Choice of Data Elements that specify either the Framework or the type of entity attempting to access the Framework.

	Tag Element Type	
	TpDomainIDType	

Tag Element Value	Choice Element Type	Choice Element Name
P_FW	TpFwID	FwID
P_CLIENT_APPLICATION	TpClientAppID	ClientAppID
P_ENT_OP	TpEntOpID	EntOpID
P_SERVICE_INSTANCE	TpServiceInstanceID	ServiceID (See Note)
P_SERVICE_SUPPLIER	TpServiceSupplierID	ServiceSupplierID

Note: The Choice Element Name ServiceID of TpDomainID refers to a service instance.

10.1.4 TpDomainIDType

Defines either the Framework or the type of entity attempting to access the Framework.

Name	Value	Description
P_FW	0	The Framework
P_CLIENT_APPLICATION	1	A client application
P_ENT_OP	2	An enterprise operator
P_SERVICE_INSTANCE	3	A service instance
P_SERVICE_SUPPLIER	4	A service supplier

10.1.5 TpEntOpID

This data type is identical to TpString and is defined as a string of characters that identifies an enterprise operator. In conjunction with the application it uniquely identifies the enterprise operator which uses a particular OSA Service Capability Feature (SCF).

10.1.6 TpPropertyName

This data type is identical to TpString. It is the name of a generic 'property'.

10.1.7 TpPropertyValue

This data type is identical to TpString. It is the value (or the list of values) associated with a generic 'property'.

10.1.8 TpProperty

This data type is a Sequence of Data Elements which describes a generic 'property'. It is a structured data type consisting of the following {name,value} pair:

Sequence Element Name	Sequence Element Type
PropertyName	TpPropertyName
PropertyValue	TpPropertyValue

10.1.9 TpPropertyList

This data type defines a Numbered List of Data Elements of type TpProperty.

10.1.10 TpEntOpIDList

This data type defines a Numbered Set of Data Elements of type TpEntOpID.

10.1.11 TpFwID

This data type is identical to TpString and identifies the Framework.

10.1.12 TpService

This data type is a Sequence of Data Elements which describes a registered SCFs. It is a structured type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceID	TpServiceID	
ServiceDescription	TpServiceDescription	This field contains the description of the service

10.1.13 TpServiceList

This data type defines a Numbered Set of Data Elements of type TpService.

10.1.14 TpServiceDescription

This data type is a Sequence of Data Elements which describes a registered SCF. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceTypeName	TpServiceTypeName	
ServicePropertyList	TpServicePropertyList	

10.1.15 TpServiceID

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies a registered SCF interface. The string is automatically generated by the Framework.

10.1.16 TpServiceIDList

This data type defines a Numbered Set of Data Elements of type TpServiceID.

10.1.17 TpServiceInstanceID

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies an instance of a registered SCF interface. The string is automatically generated by the Framework.

10.1.18 TpServiceTypeProperty

This data type is a Sequence of Data Elements which describes a service property associated with a service type. It defines the name and mode of the service property, and also the service property type: e.g. Boolean, integer. It is similar to, but distinct from, TpServiceProperty. The latter is associated with an actual service: it defines the service property's name and mode, but also defines the list of values assigned to it.

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServiceTypePropertyMode	TpServiceTypePropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

10.1.19 TpServiceTypePropertyList

This data type defines a Numbered Set of Data Elements of type TpServiceTypeProperty.

10.1.20 TpServiceTypePropertyMode

This type defines SCF property modes.

Name	Value	Documentation
NORMAL	0	The value of the corresponding SCF property type may optionally be provided
MANDATORY	1	The value of the corresponding SCF property type shall be provided at service registration time
READONLY	2	The value of the corresponding SCF property type is optional, but once given a value it can not be modified/restricted by a service level agreement
MANDATORY_READONLY	3	The value of the corresponding SCF property type shall be provided but can not subsequently be modified/restricted by a service level agreement.

10.1.21 TpServicePropertyTypeName

This data type is identical to TpString and describes a valid SCF property type name. Valid service property type names are detailed in 10.1.

10.1.22 TpServicePropertyName

This data type is identical to TpString. It defines a valid SCF property name. The valid service property names are detailed in 10.3 and in the SCF data definitions. Additionally, service property names for proprietary service properties (used for service sub types) are possible.

10.1.23 TpServicePropertyNameList

This data type defines a Numbered Set of Data Elements of type TpServicePropertyName.

10.1.24 TpServicePropertyValue

This data type is identical to TpString and describes a valid value of a SCF property.

10.1.25 TpServicePropertyValueList

This data type defines a Numbered Set of Data Elements of type TpServicePropertyValue.

10.1.26 TpServiceProperty

This data type is a Sequence of Data Elements which describes an 'SCF property'. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServicePropertyValueList	TpServicePropertyValueList	

10.1.27 TpServicePropertyList

This data type defines a Numbered Set of Data Elements of type TpServiceProperty.

10.1.28 TpServiceSupplierID

This is an identifier for a service supplier. It is used to identify the supplier to the Framework. This data type is identical to TpString.

10.1.29 TpServiceTypeDescription

This data type is a Sequence of Data Elements which describes an SCF type. It is a structured data type. It consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceTypePropertyList	TpServiceTypePropertyList	a sequence of property name and property mode tuples associated with the SCF type
ServiceTypeNameList	TpServiceTypeNameList	the names of the super types of the associated SCF type
AvailableOrUnavailable	TpBoolean	an indication whether the SCF type is available (true) or unavailable (false)

10.1.30 TpServiceTypeName

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the type of an SCF interface. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF name
P_GENERIC_CALL_CONTROL	The name of the Generic Call Control SCF
P_MULTI_PARTY_CALL_CONTROL	The name of the MultiParty Call Control SCF
P_MULTI_MEDIA_CALL_CONTROL	The name of the MultiMedia Call Control SCF
P_CONFERENCE_CALL_CONTROL	The name of the Conference Call Control SCF
P_USER_INTERACTION	The name of the User Interaction SCFs
P_USER_INTERACTION_ADMIN	The name of the User Interaction Administration SCF
P_TERMINAL_CAPABILITIES	The name of the Terminal Capabilities SCF
P_USER_BINDING	The name of the User Binding SCF
P_USER_LOCATION	The name of the User Location SCF
P_USER_LOCATION_CAMEL	The name of the Network User Location SCF
P_USER_LOCATION_EMERGENCY	The name of the User Location Emergency SCF
P_USER_STATUS	The name of the User Status SCF
P_EXTENDED_USER_STATUS	The name of Extended User Status SCF
P_DATA_SESSION_CONTROL	The name of the Data Session Control SCF
P_GENERIC_MESSAGING	The name of the Generic Messaging SCF
P_CONNECTIVITY_MANAGER	The name of the Connectivity Manager SCF
P_CHARGING	The name of the Charging SCF
P_ACCOUNT_MANAGEMENT	The name of the Account Management SCF
P_POLICY_PROVISIONING	The name of the Policy Management provisioning SCF
P_POLICY_EVALUATION	The name of the Policy Management policy evaluation SCF
P_PAM_ACCESS	The name of PAM presentity SCF
P_PAM_EVENT_MANAGEMENT	The name of PAM watcher SCF
P_PAM_PROVISIONING	The name of PAM provisioning SCF
P_MULTI_MEDIA_MESSAGING	The name of the Multimedia Messaging SCF
P_SERVICE_BROKER	The name of the Service Broker SCF

10.1.31 TpServiceTypeNameList

This data type defines a Numbered Set of Data Elements of type TpServiceTypeName.

10.1.32 TpSubjectType

Defines the subject of a query/notification request/result.

Name	Value	Description
P_SUBJECT_UNDEFINED	0	The subject is neither the framework nor the client application
P_SUBJECT_CLIENT_APP	1	The subject is the client application
P_SUBJECT_FW	2	The subject is the framework

10.1.33 TpServiceTypePropertyValue

This data type is a [Sequence of Data Elements](#) which describes a service property associated with a service. It defines the name and mode of the service property, the service property type (e.g. Boolean, integer), and also value. It is similar to, but distinct from, TpServiceProperty. The latter does not define the modes and types and is used to register values for known service properties only.

Sequence ElementName	Sequence ElementType	Documentation
ServicePropertyName	TpServicePropertyName	The name of the service property.
ServiceTypePropertyMode	TpServiceTypePropertyMode	The mode of the service property.
ServicePropertyTypeName	TpServicePropertyTypeName	The type of the service property.
ServicePropertyValueList	TpServicePropertyValueList	The Value-list of the service property.

10.1.34 TpServiceTypePropertyValueList

This data type defines a Numbered Set of Data Elements of type TpServiceTypePropertyValue.

10.2 Event Notification Data Definitions

10.2.1 TpFwEventName

Defines the name of event being notified.

Name	Value	Description
P_EVENT_FW_NAME_UNDEFINED	0	Undefined
P_EVENT_FW_SERVICE_AVAILABLE	1	Notification of SCS(s) available
P_EVENT_FW_SERVICE_UNAVAILABLE	2	Notification of SCS(s) becoming unavailable
P_EVENT_FW_MIGRATION_SERVICE_AVAILABLE	3	Notification of a backwards compatible SCS becoming available, to which the application can migrate.
P_EVENT_FW_APP_SESSION_CREATED	4	Notification of an application<->FW access session created. (See note 1)
P_EVENT_FW_APP_SESSION_TERMINATED	5	Notification of an application<->FW access session terminated. (See note 1)
P_EVENT_FW_APP_AGREEMENT_SIGNED	6	Notification that a service agreement has been signed. (See note 1)
P_EVENT_FW_APP_AGREEMENT_ENDED	7	Notification that a service agreement has been ended/terminated. (See note 1)
NOTE: These events can only be requested by enterprise operators. If requested by any other entity then the method will throw the P_INVALID_CRITERIA exception.		

10.2.2 TpFwEventCriteria

Defines the Tagged Choice of Data Elements that specifies the criteria for an event notification to be generated.

Tag Element Type	
	TpFwEventName

Tag Element Value	Choice Element Type	Choice Element Name
P_EVENT_FW_NAME_UNDEFINED	TpString	EventNameUndefined
P_EVENT_FW_SERVICE_AVAILABLE	TpServiceTypeNameList	ServiceTypeNameList
P_EVENT_FW_SERVICE_UNAVAILABLE	TpServiceTypeNameList	UnavailableServiceTypeNameList
P_EVENT_FW_MIGRATION_SERVICE_AVAILABLE	TpServiceTypeNameList	CompatibleServiceTypeNameList
P_EVENT_FW_APP_SESSION_CREATED	TpClientAppIDList	SessionCreatedList
P_EVENT_FW_APP_SESSION_TERMINATED	TpClientAppIDList	SessionTerminatedList
P_EVENT_FW_APP_AGREEMENT_SIGNED	TpClientAppIDList	AgreementSignedList
P_EVENT_FW_APP_AGREEMENT_ENDED	TpClientAppIDList	AgreementEndedList

10.2.3 TpFwEventInfo

Defines the Tagged Choice of Data Elements that specifies the information returned to the client in an event notification.

Tag Element Type	
	TpFwEventName

Tag Element Value	Choice Element Type	Choice Element Name
P_EVENT_FW_NAME_UNDEFINED	TpString	EventNameUndefined
P_EVENT_FW_SERVICE_AVAILABLE	TpServiceIDList	ServiceIDList
P_EVENT_FW_SERVICE_UNAVAILABLE	TpServiceIDList	UnavailableServiceIDList
P_EVENT_FW_MIGRATION_SERVICE_AVAILABLE	TpFWMigrationServiceAvailableInfo	MigrationServiceAvailable
P_EVENT_FW_APP_SESSION_CREATED	TpClientAppID	AppSessionCreated
P_EVENT_FW_APP_SESSION_TERMINATED	TpClientAppID	AppSessionTerminated
P_EVENT_FW_APP_AGREEMENT_SIGNED	TpFwAgreementInfo	AppAgreementSigned
P_EVENT_FW_APP_AGREEMENT_ENDED	TpFwAgreementInfo	AppAgreementEnded

10.2.4 TpFwMigrationServiceAvailableInfo

Defines the information to be supplied when an SCS becomes available.

Sequence ElementName	Sequence ElementType	Documentation
ServiceType	TpServiceTypeName	Type of SCS that has become available
ServiceID	TpServiceID	ID of the SCS that has become available
CompatibleServiceID	TpServiceID	ID of the SCS with which this new SCS is compatible with.
BackwardCompatibilityLevel	TpBoolean	Specifies if the new SCS is completely backwards compatible with the currently used SCS. Value = TRUE: SCS is completely backwards compatible Value = FALSE: SCS is not completely backwards compatible. Contact the Framework operator for more information on how to migrate.
MigrationRequired	TpBoolean	Specifies if the new SCS is replacing the existing SCS Value = TRUE: new SCS is replacing the existing one - migration is required before the date/time indicated in MigrationDateAndTime field Value = FALSE: new SCS is not replacing the existing one, but is provided in addition. All migration to the new SCS, whether required or not, shall involve the application terminating the existing service agreement and signing a new one.
DataMigrated	TpBoolean	Indicates whether all the data the application set in the previous SCS (e.g. notifications) is also available in the new SCS. Value = FALSE : the new SCS has not obtained all data (e.g. notifications) related to the old SCS and the application needs to reset all the previous data. Value = TRUE: the new SCS has obtained data (e.g. notifications) related to the old SCS, the application can use the new SCS without resetting data.
MigrationDateAndTime	TpDateAndTime	Indicates the date and time before which applications shall have migrated from existing the existing SCS to this new SCS. Migration to the new SCS requires the application to terminate the existing service agreement, and sign a new one. Failure to do this by the migration date and time indicated in this field may result in the service agreement being terminated by the Framework, since the service supplier may choose to unregister the service following this date and time. The value of this parameter, if present, shall be ignored if MigrationRequired is set to FALSE
MigrationAdditionalInfo	TpMigrationAdditionalInfoSet	Contains additional migration information. This is initially provided to permit addition of information in later releases without impacting backwards compatibility.

10.2.5 TpMigrationAdditionalInfo

Defines the [Tagged Choice of Data Elements](#) that specify additional migration-related information.

	Tag Element Type	
	TpMigrationAdditionalInfoType	

Tag Element Value	Choice Element Type	Choice Element Name
P_MIGRATION_INFO_UNDEFINED	NULL	Undefined

10.2.6 TpMigrationAdditionalInfoType

Defines the type of migration-related additional information.

Name	Value	Description
P_MIGRATION_INFO_UNDEFINED	0	Undefined

10.2.7 TpMigrationAdditionalInfoSet

Defines a **Numbered Set of Data Elements** of TpMigrationAdditionalInfo.

10.2.8 TpFwAgreementInfo

Defines the **Sequence of Data Elements** that specifies the information returned to the enterprise operator application in an event notification.

Sequence Element Name	Sequence Element Type	Description
ClientApplicationID	TpClientAppID	The ID of the client application
ServiceID	TpServiceID	The ID of the service for whom the agreement was signed/terminated
ServiceContractID	TpServiceContractID	The ID of the service contract related to the agreement if available, an empty string otherwise.
ServiceProfileID	TpServiceProfileID	The ID of the service profile related to the agreement if available, an empty string otherwise.

10.3 Trust and Security Management Data Definitions

10.3.1 TpAccessType

This data type is identical to a TpString. This identifies the type of access interface requested by the client application. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Network operators can define their own access interfaces to satisfy client requirements for different types of access. These can be selected using the TpAccessType, but should be preceded by the string "SP_". The following value is defined:

String Value	Description
P_OSA_ACCESS	Access using the OSA Access Interfaces: IpAccess and IpClientAccess

10.3.2 TpAuthType

This data type is identical to a TpString. It identifies the type of authentication mechanism requested by the client. It provides Network operators and clients with the opportunity to use an alternative to the OSA API Level Authentication interface. This can for example be an implementation specific authentication mechanism, e.g. CORBA Security, or a proprietary Authentication interface supported by the Network Operator. OSA API Level Authentication is the default authentication method. Other Network operator specific capabilities may also be used, but should be preceded by the string 'SP_'. The following values are defined:

String Value	Description
P_OSA_AUTHENTICATION	Authenticate using the OSA API Level Authentication Interfaces: IpAPILevelAuthentication and IpClientAPILevelAuthentication
P_AUTHENTICATION	Authenticate using the implementation specific authentication mechanism, e.g. CORBA Security.

10.3.3 Void

10.3.4 Void

10.3.5 Void

10.3.6 TpAuthDomain

This is Sequence of Data Elements containing all the data necessary to identify a domain: the domain identifier, and a reference to the authentication interface of the domain.

Sequence Element Name	Sequence Element Type	Description
DomainID	TpDomainID	Identifies the domain for authentication. This identifier is assigned to the domain during the initial contractual agreements, and is valid during the lifetime of the contract.
AuthInterface	IpInterfaceRef	Identifies the authentication interface of the specific entity. This data element has the same lifetime as the domain authentication process, i.e. in principle a new interface reference can be provided each time a domain intends to access another.

10.3.7 TpInterfaceName

This data type is identical to a TpString, and is defined as a string of characters that identify the names of the Framework SCFs that are to be supported by the OSA API. Other Network operator specific SCFs may also be used, but should be preceded by the string "SP_". The following values are defined.

Character String Value	Description
P_DISCOVERY	The name for the Discovery interface.
P_EVENT_NOTIFICATION	The name for the Event Notification interface.
P_OAM	The name for the OA&M interface.
P_LOAD_MANAGER	The name for the Load Manager interface.
P_FAULT_MANAGER	The name for the Fault Manager interface.
P_HEARTBEAT_MANAGEMENT	The name for the Heartbeat Management interface.
P_SERVICE_AGREEMENT_MANAGEMENT	The name of the Service Agreement Management interface.
P_REGISTRATION	The name for the Service Registration interface.
P_ENT_OP_ACCOUNT_MANAGEMENT	The name for the Service Subscription: Enterprise Operator Account Management interface.
P_ENT_OP_ACCOUNT_INFO_QUERY	The name for the Service Subscription: Enterprise Operator Account Information Query interface.
P_SVC_CONTRACT_MANAGEMENT	The name for the Service Subscription: Service Contract Management interface.
P_SVC_CONTRACT_INFO_QUERY	The name for the Service Subscription: Service Contract Information Query interface.
P_CLIENT_APP_MANAGEMENT	The name for the Service Subscription: Client Application Management interface.
P_CLIENT_APP_INFO_QUERY	The name for the Service Subscription: Client Application Information Query interface.
P_SVC_PROFILE_MANAGEMENT	The name for the Service Subscription: Service Profile Management interface.
P_SVC_PROFILE_INFO_QUERY	The name for the Service Subscription: Service Profile Information Query interface.

10.3.8 TpInterfaceNameList

This data type defines a Numbered Set of Data Elements of type TpInterfaceName.

10.3.9 TpServiceToken

This data type is identical to a TpString, and identifies a selected SCF. This is a free format text token returned by the Framework, which can be signed as part of a service agreement. This will contain Network operator specific information relating to the service level agreement. The serviceToken has a limited lifetime, which is the same as the lifetime of the service agreement in normal conditions. If something goes wrong the serviceToken expires, and any method accepting the serviceToken will return an error code (P_INVALID_SERVICE_TOKEN). Service Tokens will

automatically expire if the client or Framework invokes the terminateAccess method on the other's corresponding access interface.

10.3.10 TpSignatureAndServiceMgr

This is a Sequence of Data Elements containing the digital signature of the Framework for the service agreement, and a reference to the SCF manager interface of the SCF.

Sequence Element Name	Sequence Element Type
DigitalSignature	TpOctetSet
ServiceMgrInterface	IpServiceRef

The digitalSignature contains a CMS (Cryptographic Message Syntax) object (as defined in RFC 2630) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is the agreement text given by the client application. The "external signature" construct shall not be used (i.e. the eContent field in the EncapsulatedContentInfo field shall be present and contain the agreement text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention.

The ServiceMgrInterface is a reference to the SCF manager interface for the selected SCF.

10.3.11 TpSigningAlgorithm

This data type is identical to a TpString, and is defined as a string of characters that identify the signing algorithm that shall be used. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined.

String Value	Description
NULL	An empty (NULL) string indicates no signing algorithm is required
P_MD5_RSA_512	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512-bit modulus. The signature generation follows the process and format defined in RFC 2313 (PKCS#1 v1.5). The use of this signing method is deprecated.
P_MD5_RSA_1024	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 1024-bit modulus. The signature generation follows the process and format defined in RFC 2313 (PKCS#1 v1.5). The use of this signing method is deprecated.
P_RSASSA_PKCS1_v1_5_SHA1_1024	SHA-1 is used to produce a 160-bit message digest based on the input message to be signed. RSA is then used to generate the signature value, following the process defined in section 8 of RFC 2437 and format defined in section 9.2.1 of RFC 2437. The RSA private/public key pair is using a 1024-bit modulus.
P_SHA1_DSA	SHA-1 is used to produce a 160-bit message digest based on the input message to be signed. DSA is then used to generate the signature value. The signature generation follows the process and format defined in section 7.2.2 of RFC 2459.

10.3.12 TpSigningAlgorithmCapabilityList

This data type is identical to a TpString. It is a string of multiple TpSigningAlgorithm concatenated using a comma (,) as the separation character.

10.3.13 TpAuthMechanism

This data type is identical to a TpString. It identifies an authentication mechanism to be used for API Level Authentication. The following values are defined:

String Value	Description
P_OSA_MD5	Authentication is based on the use of MD5 (RFC 1321) hashing algorithm to generate a response based on a shared secret and a challenge received via challenge() method. The capability to use this algorithm is required to be supported when using CHAP (RFC 1994) but its use is not recommended.
P_OSA_HMAC_SHA1_96	Authentication is based on the use of HMAC-SHA1 (RFC 2404) hashing algorithm to generate a response based on a shared secret and a challenge received via challenge() method.
P_OSA_HMAC_MD5_96	Authentication is based on the use of HMAC-MD5 (RFC 2403) hashing algorithm to generate a response based on a shared secret and a challenge received via challenge() method.

10.3.14 TpAuthMechanismList

This data type is identical to a TpString. It is a string of multiple TpAuthMechanism concatenated using a comma (,) as the separation character.

10.4 Integrity Management Data Definitions

10.4.1 TpActivityTestRes

This type is identical to TpString and is an implementation specific result. The values in this data type are 'Available' or 'Unavailable'.

10.4.2 TpFaultStatsRecord

This defines the set of records to be returned giving fault information for the requested time period.

Sequence Element Name	Sequence Element Type
Period	TpTimeInterval
FaultStatsSet	TpFaultStatsSet

10.4.3 TpFaultStats

This defines the sequence of data elements which provide the statistics on a per fault type basis.

Sequence Element Name	Sequence Element Type	Description
Fault	TpInterfaceFault	
Occurrences	TpInt32	The number of separate instances of this fault
MaxDuration	TpInt32	The number of seconds duration of the longest fault
TotalDuration	TpInt32	The cumulative duration (all occurrences)
NumberOfClientsAffected	TpInt32	The number of clients informed of the fault by the Fw

Occurrences is the number of separate instances of this fault during the period. MaxDuration and TotalDuration are the number of seconds duration of the longest fault and the cumulative total during the period. NumberOfClientsAffected is the number of clients informed of the fault by the Framework.

10.4.4 TpFaultStatisticsError

Defines the error code associated with a failed attempt to retrieve any fault statistics information.

Name	Value	Description
P_FAULT_INFO_ERROR_UNDEFINED	0	Undefined error
P_FAULT_INFO_UNAVAILABLE	1	Fault statistics unavailable

10.4.5 TpFaultStatsSet

This data type defines a Numbered Set of Data Elements of type TpFaultStats

10.4.6 TpActivityTestID

This data type is identical to a TpInt32, and is used as a token to match activity test requests with their results..

10.4.7 TpInterfaceFault

Defines the cause of the interface fault detected.

Name	Value	Description
INTERFACE_FAULT_UNDEFINED	0	Undefined
INTERFACE_FAULT_LOCAL_FAILURE	1	A fault in the local API software or hardware has been detected
INTERFACE_FAULT_GATEWAY_FAILURE	2	A fault in the gateway API software or hardware has been detected
INTERFACE_FAULT_PROTOCOL_ERROR	3	An error in the protocol used on the client-gateway link has been detected

10.4.8 Void

10.4.9 TpFwUnavailReason

Defines the reason why the Framework is unavailable.

Name	Value	Description
FW_UNAVAILABLE_UNDEFINED	0	Undefined
FW_UNAVAILABLE_LOCAL_FAILURE	1	The Local API software or hardware has failed
FW_UNAVAILABLE_GATEWAY_FAILURE	2	The gateway API software or hardware has failed
FW_UNAVAILABLE_OVERLOADED	3	The Framework is fully overloaded
FW_UNAVAILABLE_CLOSED	4	The Framework has closed itself (e.g. to protect from fraud or malicious attack)
FW_UNAVAILABLE_PROTOCOL_FAILURE	5	The protocol used on the client-gateway link has failed

10.4.10 TpLoadLevel

Defines the Sequence of Data Elements that specify load level values.

Name	Value	Description
LOAD_LEVEL_NORMAL	0	Normal load
LOAD_LEVEL_OVERLOAD	1	Overload
LOAD_LEVEL_SEVERE_OVERLOAD	2	Severe Overload

10.4.11 TploadThreshold

Defines the Sequence of Data Elements that specify the load threshold value. The actual load threshold value is application and SCF dependent, so is their relationship with load level.

Sequence Element Name	Sequence Element Type
LoadThreshold	TpFloat

10.4.12 TploadInitVal

Defines the Sequence of Data Elements that specify the pair of load level and associated load threshold value.

Sequence Element Name	Sequence Element Type
LoadLevel	TploadLevel
LoadThreshold	TploadThreshold

10.4.13 TploadPolicy

Defines the load balancing policy.

Sequence Element Name	Sequence Element Type
LoadPolicy	TpString

10.4.14 TploadStatistic

Defines the Sequence of Data Elements that represents a load statistic record for a specific entity (i.e. Framework, service or application) at a specific date and time.

Sequence Element Name	Sequence Element Type
LoadStatisticEntityID	TploadStatisticEntityID
TimeStamp	TpDateAndTime
LoadStatisticInfo	TploadStatisticInfo

10.4.15 TploadStatisticList

Defines a Numbered List of Data Elements of type TploadStatistic.

10.4.16 TploadStatisticData

Defines the Sequence of Data Elements that represents load statistic information.

Sequence Element Name	Sequence Element Type
LoadValue (see Note)	TpFloat
LoadLevel	TploadLevel
NOTE: LoadValue is expressed as a percentage.	

10.4.17 TpLoadStatisticEntityID

Defines the Tagged Choice of Data Elements that specify the type of entity (i.e. service, application or Framework) providing load statistics.

Tag Element Type	
	TpLoadStatisticEntityType

Tag Element Value	Choice Element Type	Choice Element Name
P_LOAD_STATISTICS_FW_TYPE	TpFwID	FrameworkID
P_LOAD_STATISTICS_SVC_TYPE	TpServiceID	ServiceID
P_LOAD_STATISTICS_APP_TYPE	TpClientAppID	ClientAppID

10.4.18 TpLoadStatisticEntityType

Defines the type of entity (i.e. service, application or Framework) supplying load statistics.

Name	Value	Description
P_LOAD_STATISTICS_FW_TYPE	0	Framework-type load statistics
P_LOAD_STATISTICS_SVC_TYPE	1	Service-type load statistics
P_LOAD_STATISTICS_APP_TYPE	2	Application-type load statistics

10.4.19 TpLoadStatisticInfo

Defines the Tagged Choice of Data Elements that specify the type of load statistic information (i.e. valid or invalid).

Tag Element Type	
	TpLoadStatisticInfoType

Tag Element Value	Choice Element Type	Choice Element Name
P_LOAD_STATISTICS_VALID	TpLoadStatisticData	LoadStatisticData
P_LOAD_STATISTICS_INVALID	TpLoadStatisticError	LoadStatisticError

10.4.20 TpLoadStatisticInfoType

Defines the type of load statistic information (i.e. valid or invalid).

Name	Value	Description
P_LOAD_STATISTICS_VALID	0	Valid load statistics
P_LOAD_STATISTICS_INVALID	1	Invalid load statistics

10.4.21 TpLoadStatisticError

Defines the error code associated with a failed attempt to retrieve any load statistics information.

Name	Value	Description
P_LOAD_INFO_ERROR_UNDEFINED	0	Undefined error
P_LOAD_INFO_UNAVAILABLE	1	Load statistics unavailable

10.4.22 TpSvcAvailStatusReason

Defines the reason detailing the change in status of Service Instance availability.

Name	Value	Description
SVC_UNAVAILABLE_UNDEFINED	0	Undefined. A permanent failure. See Note 1.
SVC_UNAVAILABLE_LOCAL_FAILURE	1	The Local API software or hardware has failed. A permanent failure. See Note 1.
SVC_UNAVAILABLE_GATEWAY_FAILURE	2	The gateway API software or hardware has failed. A permanent failure. See Note 1.
SVC_UNAVAILABLE_OVERLOADED	3	The Service Instance is fully overloaded. A temporary problem. See Note 2.
SVC_UNAVAILABLE_CLOSED	4	The Service Instance has closed itself (e.g. to protect from fraud or malicious attack). A permanent failure. See Note 1.
SVC_UNAVAILABLE_NO_RESPONSE	5	The Framework has detected that a Service Instance has failed: e.g. non-response from an activity test, failure to return heartbeats. A permanent failure. See Note 1.
SVC_UNAVAILABLE_SW_UPGRADE	6	The Service Instance is unavailable due to software upgrade or other similar maintenance. A permanent failure. See Note 1.
SVC_AVAILABLE	7	The Service has become available again
Note 1: The client application must act to reset its use of the specified service instance (using the normal mechanisms, such as the discovery and authentication interfaces, to stop use of this service instance and begin use of a different service instance). Note 2: The "expected" recovery time could be defined within the SLA.		

10.4.23 TpAppAvailStatusReason

Defines the reason detailing the change in status of Application availability.

Name	Value	Description
APP_UNAVAILABLE_UNDEFINED	0	Undefined. A permanent failure. See Note 1.
APP_UNAVAILABLE_LOCAL_FAILURE	1	A local failure in the Application has been detected. A permanent failure. See Note 1.
APP_UNAVAILABLE_REMOTE_FAILURE	2	A remote failure to the application has been detected, e.g. a database is not working. A permanent failure. See Note 1.
APP_UNAVAILABLE_OVERLOADED	3	The Application is fully overloaded. A temporary problem. See Note 2.
APP_UNAVAILABLE_CLOSED	4	The Application has closed itself (e.g. to protect from fraud or malicious attack) . A permanent failure. See Note 1.
APP_UNAVAILABLE_NO_RESPONSE	5	The Framework has detected that the application has failed: e.g. non-response from an activity test, failure to return heartbeats. A permanent failure. See Note 1.
APP_UNAVAILABLE_SW_UPGRADE	6	The Application is unavailable due to SW upgrade or other similar maintenance. A permanent failure. See Note 1.
APP_AVAILABLE	7	The Application has become available
Note 1: The client application is unable (or does not wish) to continue using the service instance. Note 2: The "expected" recovery time could be defined within the SLA.		

10.4.24 TpLoadTestID

This data type is identical to a TpInt32, and is used as a token to match load statistics requests with their results.

10.4.25 TpFaultStatsErrorList

Defines a [Numbered List of Data Elements](#) of type TpFaultStatisticsError.

10.4.26 TpFaultReqID

This data type is identical to a TpInt32, and is used as a token to match fault statistics requests with their results.

10.4.27 TpFwAvailStatusReason

Defines the reason detailing the change in status of Framework availability.

Name	Value	Description
FRAMEWORK_UNAVAILABLE_UNDEFINED	0	Undefined. A permanent failure. See Note 1.
FRAMEWORK_UNAVAILABLE_LOCAL_FAILURE	1	A local failure in the Framework has been detected. A permanent failure. See Note 1.
FRAMEWORK_UNAVAILABLE_REMOTE_FAILURE	2	A remote failure to the Framework has been detected, e.g. a database is not working. A permanent failure. See Note 1.
FRAMEWORK_UNAVAILABLE_OVERLOADED	3	The Framework is fully overloaded. A temporary problem. See Note 2.
FRAMEWORK_UNAVAILABLE_CLOSED	4	The Framework has closed itself (e.g. to protect from fraud or malicious attack). A permanent failure. See Note 1.
FRAMEWORK_UNAVAILABLE_PROTOCOL_FAILURE	5	The Framework has detected that the protocol used between client and framework has failed. A permanent failure. See Note 1.
FRAMEWORK_UNAVAILABLE_SW_UPGRADE	6	The Framework is unavailable due to SW upgrade or other similar maintenance. A permanent failure. See Note 1.
FRAMEWORK_AVAILABLE	7	The Framework has become available
Note 1: The Framework is unable (or does not wish) to continue using the client or service instance.		
Note 2: The 'expected' recovery time could be part of the Framework's local policies.		

10.5 Service Subscription Data Definitions

10.5.1 TpPropertyName

This data type is identical to TpString. It is the name of a generic 'property'.

10.5.2 TpPropertyValue

This data type is identical to TpString. It is the value (or the list of values) associated with a generic 'property'.

10.5.3 TpProperty

This data type is a Sequence of Data Elements which describes a generic 'property'. It is a structured data type consisting of the following {name,value} pair:

Sequence Element Name	Sequence Element Type
PropertyName	TpPropertyName
PropertyValue	TpPropertyValue

10.5.4 TpPropertyList

This data type defines a Numbered List of Data Elements of type TpProperty.

10.5.5 TpEntOpProperties

This data type is of type TpPropertyList. It identifies the list of properties associated with an enterprise operator: e.g. name, organisation, address, phone, e-mail, fax, payment method (credit card, bank account).

10.5.6 TpEntOp

This data type is a **Sequence of Data Elements** which describes an enterprise operator. It is a structured data type, consisting of a unique 'enterprise operator ID' and a list of 'enterprise operator properties', as follows:

Sequence Element Name	Sequence Element Type
EntOpID	TpEntOpID
EntOpProperties	TpEntOpProperties

10.5.7 TpServiceContractID

This data type is identical to TpString. It uniquely identifies the contract, between an enterprise operator and the Framework, for the use of an OSAservice by the enterprise.

10.5.8 TpServiceContractIDList

This data type defines a **Numbered List of Data Elements** of type TpServiceContractID.

10.5.9 TpPersonName

This data type is identical to TpString. It is the name of a generic 'person'.

10.5.10 TpPostalAddress

This data type is identical to TpString. It is the mailing address of a generic 'person'.

10.5.11 TpTelephoneNumber

This data type is identical to TpString. It is the telephone number of a generic 'person'.

10.5.12 TpEmail

This data type is identical to TpString. It is the email address of a generic 'person'.

10.5.13 TpHomePage

This data type is identical to TpString. It is the web address of a generic 'person'.

10.5.14 TpPersonProperties

This data type is of type TpPropertyList. It identifies the list of additional properties, other than those listed above, that can be associated with a generic 'person'.

10.5.15 TpPerson

This data type is a **Sequence of Data Elements** which describes a generic 'person': e.g. a billing contact, a service requestor. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
PersonName	TpPersonName
PostalAddress	TpPostalAddress
TelephoneNumber	TpTelephoneNumber
Email	TpEmail
HomePage	TpHomePage
PersonProperties	TpPersonProperties

10.5.16 TpServiceStartDate

This is of type **TpDateAndTime**. It identifies the contractual start date and time for the use of an OSA service by an enterprise or an enterprise Subscription Assignment Group (SAG).

10.5.17 TpServiceEndDate

This is of type **TpDateAndTime**. It identifies the contractual end date and time for the use of an OSA service by an enterprise or an enterprise Subscription Assignment Group (SAG).

10.5.18 TpServiceRequestor

This is of type **TpPerson**. It identifies the enterprise person requesting use of an OSA service: e.g. the enterprise operator.

10.5.19 TpBillingContact

This is of type **TpPerson**. It identifies the enterprise person responsible for billing issues associated with an enterprise's use of an OSA service.

10.5.20 TpServiceSubscriptionProperties

This is of type **TpServicePropertyList**. It specifies a subset of all available service properties and service property values that apply to an enterprise's use of an OSA service.

10.5.21 TpServiceContract

This data type is a **Sequence of Data Elements** which represents a service contract. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceContractID	TpServiceContractID
ServiceContractDescription	TpServiceContractDescription

10.5.22 TpServiceContractDescription

This data type is a [Sequence of Data Elements](#) which describes a service contract. This contract should conform to a previously negotiated high-level agreement (regarding OSA services, their usage and the price, etc.), if any, between the enterprise operator and the framework operator. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceRequestor	TpServiceRequestor
BillingContact	TpBillingContact
ServiceStartDate	TpServiceStartDate
ServiceEndDate	TpServiceEndDate
ServiceTypeName	TpServiceTypeName
ServiceID	TpServiceID
ServiceSubscriptionProperties	TpServiceSubscriptionProperties
InUse	TpBoolean (See note)
Note:	The InUse flag indicates if the contract, or one of its associated profiles, is currently in use by a service instance and will be returned in describeServiceContract(). This flag will be ignored if it is passed in to createServiceContract().

10.5.23 TpClientAppProperties

This is of type TpPropertyList. The client application properties is a list of {name,value} pairs, for bilateral agreement between the enterprise operator and the Framework.

10.5.24 TpClientAppDescription

This data type is a [Sequence of Data Elements](#) which describes an enterprise client application. It is a structured data type, consisting of a unique 'client application ID', password and a list of 'client application properties':

Sequence Element Name	Sequence Element Type
ClientAppID	TpClientAppID
ClientAppProperties	TpClientAppProperties
HasAccessSession	TpBoolean (See note 1)
HasServiceInstances	TpBoolean(See note 2)
Note 1:	The HasAccessSession flag indicates if the client application currently has an access session active with the framework and will be returned in describeClientApp(). This flag will be ignored if it is passed in to createClientApp().
Note 2:	The HasServiceInstances flag indicates if the client application currently has service instances active and will be returned in describeClientApp(). This flag will be ignored if it is passed in to createClientApp(). This flag must be false if hasAccessSession is false.

10.5.25 TpSagID

This data type is identical to TpString. It uniquely identifies a Subscription Assignment Group (SAG) of client applications within an enterprise.

10.5.26 TpSagIDList

This data type defines a [Numbered List of Data Elements](#) of type TpSagID.

10.5.27 TpSagDescription

This data type is identical to TpString. It describes a SAG: e.g. a list of identifiers of the constituent client applications, the purpose of the 'grouping'.

10.5.28 TpSag

This data type is a *Sequence of Data Elements* which describes a Subscription Assignment Group (SAG) of client applications within an enterprise. It is a structured data type consisting of a unique SAG ID and a description:

Sequence Element Name	Sequence Element Type
SagID	TpSagID
SagDescription	TpSagDescription

10.5.29 TpServiceProfileID

This data type is identical to TpString. It uniquely identifies the service profile, which further constrains how an enterprise SAG uses an OSA service.

10.5.30 TpServiceProfileIDList

This data type defines a *Numbered List of Data Elements* of type TpServiceProfileID.

10.5.31 TpServiceProfile

This data type is a *Sequence of Data Elements* which represents a Service Profile. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceProfileID	TpServiceProfileID
ServiceProfileDescription	TpServiceProfileDescription

10.5.32 TpServiceProfileDescription

This data type is a [Sequence of Data Elements](#) which describes a Service Profile. A service contract contains one or more Service Profiles, one for each SAG in the enterprise operator domain. A service profile is a restriction of the service contract in order to provide restricted service features to a SAG. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceContractID	TpServiceContractID
ServiceStartDate	TpServiceStartDate
ServiceEndDate	TpServiceEndDate
ServiceTypeName	TpServiceTypeName (See note 1)
ServiceSubscriptionProperties	TpServiceSubscriptionProperties
InUse	TpBoolean (See note 2)
ServiceID	TpServiceID (See note 3)
Note 1:	When the Framework returns a TpServiceProfileDescription to the enterprise operator, it should set the ServiceTypeName field to the same value as the corresponding field of the service contract; When the enterprise operator passes a TpServiceProfileDescription to the Framework, the Framework should ignore the value sent in the ServiceTypeName field to ensure interoperability; The enterprise operator should be required to set the ServiceTypeName field to the correct value when passing a TpServiceProfileDescription to the Framework.
Note 2:	The InUse flag indicates if the profile is currently in use by a service instance and will be returned in describeServiceProfile(). This flag will be ignored if it is passed in to createServiceProfile().
Note 3:	The ServiceID field is used to restrict a service type-based service contract to a specific service. When the TpServiceProfileDescription is passed to the Framework by an enterprise operator, the Framework should ensure that the ServiceID field, if not empty, contains a service which is of the service type specified in the service contract. If the corresponding contract is for a service ID then the Framework should ignore this field. When a TpServiceProfileDescription is returned to the enterprise operator, the contents of this field will depend on the associated service contract. If the contract is for a service ID, then this field should be populated with the correct value. If the contract is for a service type, and the profile is restricted to a specific service ID then this field should be populated with the correct value. Otherwise, it should contain an empty string.

10.5.33 TpSagProfilePair

This data type is a Sequence of Data Elements which describes a pair of aSAG and a Service Profile. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
Sag	TpSagID
ServiceProfile	TpServiceProfileID

10.5.34 TpAddSagMembersConflict

This data type is a Sequence of Data Elements which describes a conflict that may occur when client applications are added to a SAG - see method addSagMembers(). This happens, when a client application is assigned to a service twice.

The AlreadyAssignedSagProfilePair describes the SAG and the service profile through which the client application is already assigned to the service. It includes the current service profile. The ConflictGeneratingSagProfilePair describes another SAG, to which the client application should be added, and the corresponding service profile, through which the client application is also connected to this service. This creates a conflict, as there may exist only a single service profile for each service.

The TpAddSagMembersConflict is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ClientApplication	TpClientAppID
ConflictGeneratingSagProfilePair	TpSagProfilePair
AlreadyAssignedSagProfilePair	TpSagProfilePair
Service	TpServiceID

10.5.35 TpAddSagMembersConflictList

This data type defines a [Numbered List of Data Elements](#) of type TpAddSagMembersConflict.

10.5.36 TpAssignSagToServiceProfileConflict

This data type is a Sequence of Data Elements which describes a conflict that may occur when a SAG is assigned to a Service Profile - see method assign().

The AlreadyAssignedSagProfilePair describes the SAG and the service profile through which the client application is already assigned to the service.

The TpAssignSagToServiceProfileConflict is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ClientApplication	TpClientAppID
AlreadyAssignedSagProfilePair	TpSagProfilePair
Service	TpServiceID

10.5.37 TpAssignSagToServiceProfileConflictList

This data type defines a [Numbered List of Data Elements](#) of type TpAssignSagToServiceProfileConflict.

11 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_ACCESS_DENIED	The client is not currently authenticated with the framework
P_DUPLICATE_PROPERTY_NAME	A duplicate property name has been received
P_ILLEGAL_SERVICE_ID	Illegal Service ID
P_ILLEGAL_SERVICE_TYPE	Illegal Service Type
P_INVALID_ACCESS_TYPE	The framework does not support the type of access interface requested by the client.
P_INVALID_ACTIVITY_TEST_ID	ID does not correspond to a valid activity test request
P_INVALID_ADDITION_TO_SAG	A client application cannot be added to the SAG because this would imply that the client application has two concurrent service profiles at a particular moment in time for a particular service.
P_INVALID_AGREEMENT_TEXT	Invalid agreement text
P_INVALID_ENCRYPTION_CAPABILITY	Invalid encryption capability
P_INVALID_AUTH_TYPE	Invalid type of authentication mechanism
P_INVALID_CLIENT_APP_ID	Invalid Client Application ID
P_INVALID_DOMAIN_ID	Invalid client ID
P_INVALID_ENT_OP_ID	Invalid Enterprise Operator ID
P_INVALID_PROPERTY	The framework does not recognise the property supplied by the client
P_INVALID_SAG_ID	Invalid Subscription Assignment Group ID
P_INVALID_SAG_TO_SERVICE_PROFILE_ASSIGNMENT	A SAG cannot be assigned to the service profile because this would imply that a client application has two concurrent service profiles at a particular moment in time for a particular service.
P_INVALID_SERVICE_CONTRACT_ID	Invalid Service Contract ID
P_INVALID_SERVICE_ID	Invalid service ID
P_INVALID_SERVICE_PROFILE_ID	Invalid service profile ID
P_INVALID_SERVICE_TOKEN	The service token has not been issued, or it has expired.
P_INVALID_SERVICE_TYPE	Invalid Service Type
P_INVALID_SIGNATURE	Invalid digital signature
P_INVALID_SIGNING_ALGORITHM	Invalid signing algorithm
P_MISSING_MANDATORY_PROPERTY	Mandatory Property Missing
P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY	No encryption mechanism, which is acceptable to the framework, is supported by the client
P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM	No authentication mechanism, which is acceptable to the framework, is supported by the client
P_NO_ACCEPTABLE_SIGNING_ALGORITHM	No signing algorithm, which is acceptable to the framework, is supported by the client
P_PROPERTY_TYPE_MISMATCH	Property Type Mismatch
P_SERVICE_ACCESS_DENIED	The client application is not allowed to access this service.
P_SERVICE_NOT_ENABLED	The service ID does not correspond to a service that has been enabled
P_SERVICE_TYPE_UNAVAILABLE	The service type is not available according to the Framework.
P_UNKNOWN_SERVICE_ID	Unknown Service ID
P_UNKNOWN_SERVICE_TYPE	Unknown Service Type

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
ExtraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

Annex A (normative): OMG IDL Description of Framework

The OMG IDL representation of this interface specification is contained in text files (fw_data.idl, fw_if_access.idl, fw_if_app.idl, fw_if_service.idl contained in archive 2919803V710IDL.ZIP) which accompany the present document.

Annex B (informative): W3C WSDL Description of Framework

The W3C WSDL representation of this interface specification is contained in zip file 2919803V710WSDL.ZIP, which accompanies the present document.

Annex C (informative): Java™ API Description of the Framework

The Java™ API realisation of this interface specification is produced in accordance with the Java™ Realisation rules defined in Part 1 of this specification series. These rules aim to deliver for Java™, a developer API, provided as a realisation, supporting a Java™ API that represents the UML specifications. The rules support the production of both J2SE™ and J2EE™ versions of the API from the common UML specifications.

The J2SE™ representation of this interface specification is provided as Java™, contained in archive 2919803V710J2SE.ZIP that accompanies the present document.

The J2EE™ representation of this interface specification is provided as Java™, contained in archive 2919803V710J2EE.ZIP that accompanies the present document.

Annex D (informative): Description of the Framework for 3GPP2 cdma2000 networks

This annex is intended to define the OSA API Stage 3 interface definitions and it provides the complete OSA specifications for cdma2000-based systems. It is an extension of OSA API specifications capabilities to enable operation in cdma2000 systems environment. They are in alignment with 3GPP2 Stage 1 requirements and Stage 2 architecture defined in

- [1] 3GPP2 P.S0001-B: "Wireless IP Network Standard", Version 1.0, September 2000;
- [2] 3GPP2 S.R0037-0: "IP Network Architecture Model for cdma2000 Spread Spectrum Systems", Version 2.0, May 14, 2002;
- [3] 3GPP2 X.S0013: "All-IP Core Network Multimedia Domain", December 2003.

These requirements are expressed as additions to and/or exclusions from the 3GPP Release 7 specification. The information given here is to be used by developers in 3GPP2 cdma2000 network architecture to interpret the 3GPP OSA specifications.

D.1 General Exceptions

The term UMTS is not applicable for the cdma2000 family of standards. Nevertheless the term UMTS is used in 3GPP TR 21.905 (Vocabulary for 3GPP Specifications) mostly in the broader sense of "3G Wireless System". If not stated otherwise there are no additions or exclusions required.

CAMEL and CAP mappings are not applicable for cdma2000 systems.

D.2 Specific Exceptions

D.2.1 Clause 1: Scope

There are no additions or exclusions.

D.2.2 Clause 2: References

Normative references on 3GPP TS 23.078 and on 3GPP TS 29.078 are not applicable for cdma2000 systems.

D.2.3 Clause 3: Definitions and abbreviations

There are no additions or exclusions.

D.2.4 Clause 4: Overview of the Framework

There are no additions or exclusions.

D.2.5 Clause 5: The Base Interface Specification

There are no additions or exclusions.

D.2.6 Clause 6: Framework Access Session API

There are no additions or exclusions.

D.2.7 Clause 7 Framework-to-Application Sequence Diagrams

There are no additions or exclusions.

D.2.8 Clause 8: Framework-to-Service API

There are no additions or exclusions.

D.2.9 Clause 9: Service Properties

Since CAMEL protocol is not applicable for cdma2000 systems, an SCS shall indicate support for the CAMEL feature through service properties. For cdma2000 systems the CAMEL service properties shall be disabled (CAMEL shall be turned always off in the case of the 3GPP2 networks; e.g.: UserLocationCamel shall be set to false).

D.2.10 Clause 10: Data Definitions

There are no additions. P_USER_LOCATION_CAMELvalue of TpServiceTypeName is not required to be supported in the 3GPP2 networks.

D.2.11 Clause 11: Exception Classes

There are no additions or exclusions.

D.2.12 Annex A (normative): OMG IDL Description of the Framework

There are no additions or exclusions.

D.2.13 Annex B (informative): W3C WSDL Description of the Framework

There are no additions or exclusions.

D.2.14 Annex C (informative): Java™ API Description of the Framework

There are no additions or exclusions.

Annex E (informative): Change history

Change history							
Date	TSG #	TSG Doc.	CR	Rev	Subject/Comment	Old	New
Jun 2006	CT_32	CP-060205	0133	--	Remove deprecated items from Trust and Security Management interfaces	6.6.1	7.0.0
Jun 2006	CT_32	CP-060205	0134	--	Remove deprecated items from Integrity Management: Fault and Load Management	6.6.1	7.0.0
Sep 2006	--	--	--	--	Added missing code attachments J2EE and J2SE.	7.0.0	7.0.1
Dec 2006	CT_34	CP-060721	0136	1	Remove unintended limitation on the support of regular expressions within TpAddressRange	7.0.1	7.1.0
Dec 2006	CT_34	CP-060596	0137	--	Add TpServiceTypeName to include name of Service Broker SCF	7.0.1	7.1.0

History

Document history		
V7.1.0	June 2007	Publication